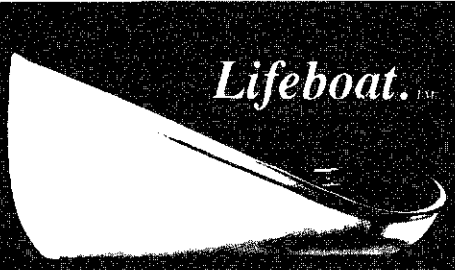
Vol. 3 No. 4 July/August 1985 $4.00

programmer's journal

The Resource Journal for IBM PC Programmers

BULK RATE
U.S. POSTAGE
PAID
EUGENE, OR 97401
PERMIT NO. 668

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Lifeboat.™

C is the language.
Lifeboat™ is the source.

Productivity Tools from the Leading Publisher of C Programs.™

The Lattice® C Compiler

The cornerstone of a program is its compiler; it can make the difference between a good program and a great one. The Lattice C compiler features:

- Full compatibility with Kernighan and Ritchie's standards
- Four memory model options for control and versatility
- Automatic sensing and use of the 8087 math chip
- Choose from the widest selection of add-on options
- Renowned for speed and code quality
- Superior quality documentation

"Lattice C produces remarkable code...the documentation sets such a high standard that others don't even come close...in the top category for its quick compilation and execution time and consistent reliability."

Ralph A. Phraner, *Byte* Magazine

Lattice Library source code also available.

Language Utilities

Pfix 86/Pfix 86 Plus — dynamic and symbolic debuggers respectively, these provide multiple-window debugging with breakpointing capability.
Plink 86 — a two-pass overlay linkage editor that helps solve memory problems.
Text Management Utilities — includes GREP (searches files for patterns), DIFF (differential text file comparator), and more.
LMK (UNIX "make") — automates the construction of large multi-module products.
Curses — lets you write programs with full screen output transportable among all UNIX, XENIX and PC-DOS systems without changing your source code.
BASTOC — translates MBASIC or CBASIC source code directly to Lattice C source code.
C Cross Reference Generator — examines your C source modules and produces a listing of each symbol and where it is referenced.

Editors

Pmate — a customizable full screen text editor featuring its own powerful macro command language.
ES/P for C — C program entry with automatic syntax checking and formatting.
VEDIT — an easy-to-use word processor for use with V-PRINT.
V-PRINT — a print formatting companion for VEDIT.
CVUE — a full-screen editor that offers an easy way to use command structure.
EMACS — a full screen multi window text editor.
Fast/C — speeds up the cycle of edit-compile-debug-edit-recompile.

Graphics and Screen Design

HALO — one of the industry's standard graphics development packages. Over 150 graphics commands including line, arc, box, circle and ellipse primitives. The **10 Fontpack** is also available.
Panel — a screen formatter and data entry aid.
Lattice Window — a library of subroutines allowing design of windows.

Functions

C-Food Smorgasbord — a tasty selection of utility functions for Lattice C programmers; includes a binary coded decimal arithmetic package, level 0 I/O functions, a Terminal Independence Package, and more.
Float-87 — supports the 8087 math chip to boost the speed of floating-point calculations.
The Greenleaf Functions — a comprehensive library of over 200 routines.
The Greenleaf Comm Library — an easy-to-use asynchronous communications library.
C Power Packs — sets of functions useful for a wide variety of applications.
BASIC C — This library is a simple bridge from IBM BASIC to C.

Database Record Managers

Phact — a database record manager library of C language functions, used in the creation and manipulation of large and small databases.
Btrieve — a sophisticated file management system designed for developing applications under PC-DOS. Data can be instantly retrieved by key value.
FABS — a Fast Access Btree Structure function library designed for rapid, keyed access to data files using multipath structures.
Autosort — a fast sort/merge utility.
Lattice dB-C ISAM — a library of C functions that enables you to create and access dBase format database files.

Cross-Compilers

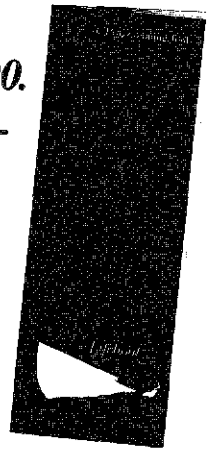
For programmers active in both micro and mini environments we provide advanced cross-compilers which product Intel 8086 object modules. All were developed to be as functional — and reliable — as the native compilers. They are available for the following systems:

VAX/VMS, VAX/UNIX, 68K/UNIX-S, 68K/UNIX-L

Also, we have available:
Z80 Cross-Compiler for MS- and PC-DOS — produces Z80 object modules in the Microsoft relocatable format.

New Products

Run/C — finally, a C interpreter for all levels of C Programmers.
C Sprite — a symbolic debugger with breakpoint capability.



Call LIFEBOAT: 1-800-847-7078. In NY, 1-212-860-0300.

YES! Please rush me the latest FREE Lifeboat™ catalog of C products.

*Name*_____ *Title*_____

*Company Name*_____ *Business Phone*_____

*Address*_____

Please check one of the following categories:

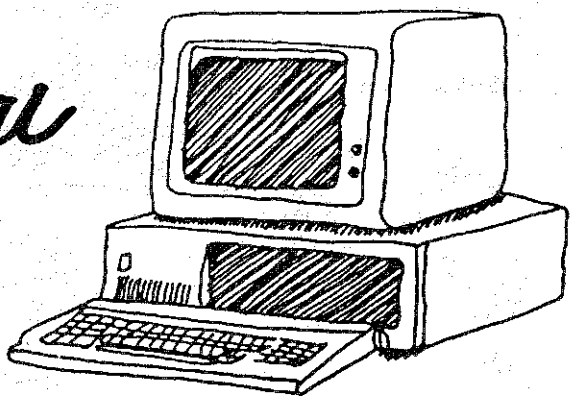
☐ *Dealer/Distributor* ☐ *End User* ☐ *Other* _____

Return Coupon to: Lifeboat™ Associates
1651 Third Avenue, New York, NY 10128

© 1985 Lifeboat Associates

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

programmer's journal



Volume 3, Number 4 July/August 1985

CONTENTS

HOUSEKEEPING 2

LETTERS 3

PJ FORUM: MORE UNIX STUFF by FNE 5
Excerpts from DTACK GROUNDED: The 68000/32081 Systems Journal

REDIRECTIONS: C PROGRAMMING BOOKS survey by Frank D. Greco 13
The Madonnas and the Twisted Sisters

C: C CODE EXCERPTS by Ron Moak 21
Polish your C programs with fast assembly routines

ASM: HOW TO CURE THE "DREADED LOCKED KEYBOARD" CONDITION by John W. Jackson 23
Bringing your computer back from Never-neverland

PASCAL: CALLING MS-DOS FROM TURBO PASCAL by Tom Swan 30
See how simple it is to exit and enter your Pascal programs

UTILITIES: DOS 2.X SORT AND FIND FILTERS by Bernard H. Robinson, Jr. 33
Professional uses for DOS filter commands

TOOLS: THE SET TOOLS review by Bill Salkin 39
Programmer's utilities that really work

MISC. 47

AD INDEX 48

ISSN# 0747-5861

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

HOUSEKEEPING

Whenever I am asked to name a piece of truly top quality software, I unhesitatingly answer WordProof. It is fast, friendly, and flawless. By any definition you choose to use, it is a classic work of art. The "less is better" philosophy has found its holy guru in WordProof. Just like VisiCalc, WordProof can almost seduce us into buying a machine just to appreciate its virtues.

Another piece of software that has recently appeared further expands the dimension of computer usage. Not that this particular product would make you run out and buy an IBM PC, but it would leave a burning desire in you to encourage a friend to buy one. Calling this wonderful little program top-drawer would be a misnomer; a more appropriate designation would be bottom-drawer — used in a nevertheless friendly way. The software I'm referring to is called INSULTS.

What you do is this: Sneak into your friend's computer cubicle, run INSULTS, and tell the program how many insults you'd like your "friend" to receive. The program then displays the normal DOS prompt, and once your buddy hits any key, he has a screen full of disgustingly creative phrases like

You low budget truckload of crumbly zit cheese

or perhaps

You dim-witted blob of synthetic rat boogers.

The program can generate over two million distinctive insults, enough for you and a large number of friends. It is fascinating to page through several screensful and imagine the kind of subroutine that generates the combinations. INSULTS is definitely a breath of fresh air in the software biz, if you'll let me call it that.

For a Freeware copy, send a diskette, etc., to

Modern Advisory Institute
P.O. Box 11632
Salt Lake City, UT 84147

They also have a more advanced version that includes additional pranks. (I wish they supplied a "to the printer" option that would send the reams of verbal gymnastics to my dot-matrix.) Well, send for a copy and have fun.

There are a number of highly useful articles and software routines in this issue of *PJ*. Frank Greco's C book reviews contain a lot of valuable C tips — don't miss them.

—Greg □

programmer's journal

Publisher:	Elizabeth Oakley
Editor:	Greg Estes
Managing Editor:	Erin O'Connor
Advertising:	Ralph Bentley
Production:	Ken Petersen
Circulation/Promotion:	Robert Tappan
Cover Design:	Nicole Harris
Illustrations:	Julia King
Contributing Writers:	FNE Frank D. Greco John W. Jackson Rex Jaeschke Ron Moak Bernard H. Robinson, Jr. Bill Salkin Tom Swan

Published by:	Oakley Publishing Company
Business and Editorial Offices:	Programmer's Journal 1897 Garden Ave. P.O. Box 30160 Eugene, OR 97403 (503) 484-2162
Subscription Inquiries:	Programmer's Journal P.O. Box 3000 Dept. EE Denville, NJ 07834 (503) 484-2162
Postmaster send address changes to:	*Programmer's Journal* *P.O. Box 3000* *Dept. EE* *Denville, NJ 07834*

Subscriptions: Bi-monthly rates are $24 U.S. ($30 in Canada, and $40 elsewhere)

IBM and Personal Computer are trademarks of International Business Machines, Armonk, New York

Published bi-monthly by
Oakley Publishing Company

Copyright 1985

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

SCAN WITHOUT GOTO

Congratulations on *PJ*. In just a few short years, the value of your magazine has risen quite a bit. I particularly enjoyed your latest cover story "Scanning in Turbo Pascal." However, the author issued a challenge that I couldn't resist, claiming that his SCAN routine contained a GOTO because it was the most efficient way he could find, and then adding that he'd like to see a faster version *without* GOTOs.

As an instructor in Pascal, I'm sorta touchy about GOTOs. Anyone using them in my class better have a fairly good reason. Usually, a REPEAT . . . UNTIL loop does the trick faster and more clearly. While reading the article on the subway, I came up with a version that is 20% faster without GOTOs. The routine is faster for three reasons.

The first and second reasons are that major switch points — whether the opcode is an "equals" or "not equals" type, and whether the search direction is positive or negative — are retested in every loop iteration, when they should be tested only once per function call. The last reason is that a REPEAT . . . UNTIL loop is slightly faster than the GOTO out of the loop.

Below is my version of SCAN. It produced 313 bytes of object code and is thus larger than the original version, which produces 232 bytes of object code. However, it is 17 to 19 percent faster than the original SCAN.

Thanks for publishing the article on SCAN. I'm finding it a very useful routine. Keep the Turbo stuff coming!

```
Function Scan(Limit:integer;Opcode,ch:char;var source):integer;
(*  Performs scan function as specified in PJ vol 3 number 2,
    BUT 1) without GOTOs and 2) faster.

    Main sources of improvement:
    1) case to test opcode is only done ONCE per function call.
    2) simple REPEAT...UNTIL loop executes faster than WHILE.
    3) increment/decrement choice only done ONCE per function
       call.
*)

type c_array = array[0..maxint] of char;
var i,j,k,Increment:integer;
    a: c array absolute source;

BEGIN

(* Initialize increment to save many CASEs later *)
   i:=0;
   IF Limit<0 THEN Increment:=-1 ELSE Increment:=1;

(* Execute this part of the routine if checking for equality *)

  IF Opcode='=' THEN
  BEGIN
    (* Test first byte separately *)
    IF a[0]=ch THEN scan:=0 ELSE

    BEGIN
      REPEAT
        i:=i+Increment
      UNTIL (a[i]=ch) or (i=Limit);
      scan:=i
    END

  END (* clause if opcode = '=' *) ELSE

(* Execute this part of the routine if checking for equality *)

  BEGIN  (* if opcode = '#' *)
    (* Test first byte separately *)
    If a[0]<>ch THEN scan:=0 ELSE
    BEGIN
      REPEAT
        i:=i+Increment
      UNTIL (a[i]<>ch) or (i=Limit);
      scan:=i;
    END
  END;
END;
```

Mark Minasi
Arlington, VA 22205

APL★PLUS® PC SYSTEM

Imagine a complete language processor that allows you to develop programs 4 to 10 times faster than you can with BASIC or Pascal.

Plus, all you need to develop business applications:
- a complete implementation of APL
- full screen editing
- a built-in terminal emulator
- a file system
- and report formatting.

Built in functions let you eliminate such time consuming programming chores as sorting, matrix inversions, and string searching.

STSC's APL★PLUS PC System does all this and more. You can integrate to non-APL programs and data files. And APL programs developed on your PC will run on other desktops and minis and mainframes.

Our complete package price is $595. Major credit cards are accepted. Call us today and we'll send you our free Convincer Kit. Contact your local dealer or call **800-592-0050** (in Maryland, call **301-984-5123**).

The APL★PLUS PC System is available for the IBM PC and compatibles, Wang Professional, and DEC Rainbow 100 operating under MS DOS with 192K of RAM memory. A character generator ROM or software is included.

APL★PLUS is a registered trademark of STSC, Inc.
IBM, WANG and DEC are trademarks of International Business Machines, Wang Laboratories, and Digital Equipment Corporation respectively.

Problem-solving at the speed of thought.

STSC
A CONTEL COMPANY

STSC, Inc.
2115 E. Jefferson St.
Rockville, Md. 20852

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

1	2
3	4

Have we got your number?

Is Programmer's Journal reaching you at the correct address? Please notify us of any address change or correction.

Is it nearing time to renew? Your Programmer's Journal mailing label shows when your subscription expires. Renew early to avoid any interruption in your subscription service!

Programmer's Journal, P.O. Box 3000, Dept. EE, Denville, NJ 07834

Tools for the Programmer from Blaise Computing

Save Up To $130 On These Special Offers!

TOOLS & TOOLS 2
For C or Pascal
For a limited time, pick up both packages and save $50 off our regular list price. The C version comes with libraries for the Lattice, Computer Innovations and Microsoft (version 2.03 and 3.00) compilers. The Pascal version supports IBM and Microsoft Pascal. **$175.**

VIEW MANAGER With Source
All libraries are included. Please specify C or Pascal. Regular $425. Save $130. **$295**

FOR C AND PASCAL PROGRAMMERS:

FOR THE TURBO PASCAL PROGRAMMER:

PACKAGES FOR ALL PROGRAMMERS:

TO ORDER, call Blaise Computing Inc. at (415) 540-5441

Famous Computer Personalities # 49: Leri "Mocha-Mocha" Stanerson, arriving for his new job as chief programmer in his favorite suit, realizes he has created the first parody error.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

MORE UNIX STUFF

Excerpts from *DTACK GROUNDED*
The 68000/32081 Systems Journal

by FNE, © 1984 by Digital Acoustics, Inc.

AT&T decided to hog the profits on System V, so independent OS vendors are avoiding it like the plague. Pay attention now: "[AT&T] changed their licensing agreement so that you couldn't pyramid them. They got restrictive. They said the company that remits the royalties to AT&T must own all the sources its sublicensees distribute. This has discouraged systems houses from going to a company like Microsoft, because they would have to hand over their source codes." That was Robert Swarz, president of Mark Williams (the *COHERENT* folk), quoted in Jan 21 *Electronic News* (p. 34).

So what does this mean? Well, you will recall that Intel (they make microprocessors, remember?) had adopted Microsoft's XENIX as their (Intel's) internal standard operating system. When Intel decided to follow AT&T's lead and implement UNIX System V on the 80286, they naturally approached Microsoft to do the job for them. Heck, everybody knows that XENIX is really UNIX, right? Uh, wrong. AT&T had set the rules such that Microsoft could not 'leverage' off XENIX — in other words, having written 'UNIX' once, Microsoft would have to write UNIX all over again from scratch. Microsoft looked at its existing work load and politely declined the job.

Intel was placed in the humiliating position of finding another software house to implement a SECOND UNIX (remember, XENIX is really UNIX, right?), this one incompatible with its internal house standard, for its 80286. Wunnerful, wunnerful! Digital Reasearch, the CP/M folk, reluctantly accepted the job. They have finally delivered. Late.

Pay attention again, now: John Rowley, DRI prexy, publicly asserted that "UNIX has not been an overwhelming success in the commercial marketplace." DRI then tossed out all of its ongoing work on UNIX System V. (*Electronic News* politely asserted that DRI "spun out" two groups. Don't you believe it: The word is 'toss'!)

The first group to be (ahem!) 'spun out' was the UNIX Library development group. That group has already folded for lack of funding.

The second group consists of the folks who did that 80286 System V port, who have set up shop under the name Microport Systems. A third group, which had been working on System V applications, is being quietly disbanded, as in 'here is your pink slip.' The total amount of money DRI has invested in any of these groups is apparently zero. Continuing to quote *EN*:

"With the move, DRI said it is abandoning any future System V porting projects and operating system-level development work.

"Of Concern to DRI was the prospect of competing directly against AT&T while abiding by AT&T's licensing terms. 'It just turned out not to be a viable thing for us to do,' Mr. Rowley noted. 'AT&T is actively marketing UNIX themselves.'

"At Microsoft, Adrian King, XENIX product director, had a similar observation, noting 'AT&T has become a fairly aggressive competitor in the marketplace.' "

So there you have it: AT&T, with its recent changes in its licensing policies, has assured that it will corner the lion's share of any monies generated by System V while simultaneously assuring that there will be precious little monies garnered because folks will look elsewhere. They already have.

For instance, just exactly who is going to port System V to Intel's forthcoming 80386. That's '386,' not '286.' Hmmm? Embarrassing, isn't it?

AT&T has shot itself in the foot again. You know, with foul-ups like this, we should not be surprised that their $16,000 'Super-Micro' is five times slower than a $69.97 VIC-20. AT&T has been trying to lay off folks lately, or maybe get them to retire early. How many folks? Try middle five figures. Yep, FIVE!

HOW FAST/SLOW IS 'C'?

It has become obvious to us over the past several years that applications programs, or (worse) other programming languages which are written in 'C' have horrendously poor performance. By 'horrendously poor,' we mean in comparison to some obviously similar programs written in assembler. To refresh one's memory, let us cite four obvious, and fairly recent, examples:

1. The floating point package provided by AT&T with their 3B2/300 $16,000 'Super Turkey.' Counter-example: the floating point package in the $69.97 VIC-20.

2. WordStar 2000. Counter-example: WordStar.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

MasterFORTH

Portable programming environment

Whether you program on the **Macintosh**, the **IBM PC**, an **Apple II** series, a **CP/M** system, or the **Commodore 64**, your program will run unchanged on all the rest. If you write for yourself, MasterFORTH will protect your investment. If you write for others, it will expand your marketplace.

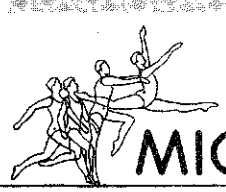
IBM™

MasterFORTH is a state-of-the-art implementation of the Forth computer language. Forth is interactive – you have immediate feedback as you program, every step of the way. Forth is fast, too, and you can use its built-in macro assembler to make it even faster. MasterFORTH's relocatable utilities, transient definitions, and headerless code let you pack a lot more program into your memory. The resident debugger lets you decompile, breakpoint, and trace your way through most programming problems. A string package, file interface, and full screen editor are all standard features.

CP/M™

MasterFORTH exactly matches the Forth-83 Standard dialect described in *Mastering Forth* by Anderson and Tracy (Brady, 1984). The standard package includes the book and over 100 pages of supplementary documentation.

MasterFORTH standard package	
Macintosh	$125
IBM PC and PC Jr. (MS DOS 2.1)	125
Apple II, II+, IIe, IIc (DOS 3.3)	100
CP/M 2.X (in several formats)	100
Commodore 64	100
Extensions	
Floating Point (1984 FVG standard)	$40
Graphics (Apple II series)	40
Module relocator (with utility sources)	60
Printed source listing (each)	35
Publications	
Mastering Forth (additional copies)	$18
Thinking Forth by Leo Brodie	16
Forth-83 International Standard	15
Rochester Bibliography, 2nd ed.	15
1984 Rochester Conference	25
1984 J1 of Forth Appl. & Res. 2(2)	15
1983 FORML Conference	25



MICROMOTION

12077 Wilshire Blvd., #506
Los Angeles, CA 90025
(213) 821-4340

3. DRI's Z-80 "Personal BASIC." Counter-example: Microsoft Z-80 BASIC.

4. Microsoft's MS-DOS FORTRAN. Counter-example: MANY competing assembly-based FORTRANs that will run circles around Microsoft's.

During the period when we have been keeping a close watch on the microcomputer scene (about 5 years now) we have never once observed an example of a 'C'-based application program (or other programming language) which was even remotely in the same performance ballpark as an equivalent assembly-based program. (The same thing can be said of OTHER high-level languages such as Pascal.) While we have obviously not been able to verify that EVERY such application or language written in 'C' is horrendously slow, we believe we have a sufficiently large statistical sample to make two observations:

1. All high level languages have a SUBSTANTIAL performance overhead with respect to assembly code written by a skilled, experienced assembly programmer.

2. Despite the obvious theoretical validity of the 90%-10% rule, such post-optimization is evidently ignored in the real world.

Let us emphasize that we reached the above conclusions from real-world observations, not from theoretical considerations.

However, the first of the two conclusions above is hotly disputed by a number of persons, including some of you readers and also including lots of folk in the academic community who have never heard of us. See, for instance, the comment on optimizing compilers by Terry Peterson in his *STRIDE FAIRE* report elsewhere in this issue. Or, the following exerpt from a recent letter from a *DTACK GROUNDED* reader:

"FNE WRONG AGAIN" DEPT:

"It is known among all experts that the reason FNE cannot find a compiler that has a 50% overhead over assembly language is that such compilers have long ago been replaced by compilers that have NO EXTRA OVERHEAD AT ALL!! What? You don't believe me? I quote from *Operating System Concepts* by Peterson and Silberschatz, p. 350: 'We now have optimizing compilers that generate code that is at least as good as hand-written assembly language programs.' Since this book is used in the teaching of operating systems to graduate students in at least two major universities, it must be true, yes? I mean, after all, what is FNE's minor opinion

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

compared to all those masters and doctorate candidates in CS? YOU WILL BELIEVE THIS! YOU WILL ALSO GO OUT AND BUY A PCjr!" Bill H., West Linn, OR.

(Bill, certain aspects of your prose style seem vaguely familiar - FNE)

One reader, who has taken a significant amount of time to educate us on the subject of the efficiency of 'C' compilers, is Daniel L. (see #39 p. 14 col. 2), who went so far as to assert that the DRI 'C' compiler has only a 1.25-1 high-level overhead. Our comments on that led to further correspondence, followed by a phone call to clarify some points. We were trying to reconcile that 1.25-1 with the glacial slowness of certain DRI languages written in 'C,' including that ill-fated "Personal Basic" which is best benchmarked using an almanac.

To our surprise, Daniel asserted something like "Yes, many applications written in 'C' are very slow even though the language 'C' — and DRI's compiler — is VERY efficient!" Daniel then proceeded to offer technical reasons why this is true. We said, HOLD IT! You just GOTTA write us a letter about this! Here, then, is Daniel's letter on this subject (and on CP/M-68K, which will be discussed separately later):

C ONLY *SEEMS* SLOW:

"I have been observing the current debate on high-level languages vs. assembler for quite some time now. I have observed that packages such as WordStar 2000 are reported as having extremely poor performance, as being excessive in size, and so on. The next thing I read is generally a diatribe starting "And it's all because this was written in 'C!' " BULLPUCKEY!! Of all the idiot things I have heard in this industry, this one takes the cake. Pay attention now: First, C is a VERY simple low-to- medium level language, so simple that even a feeble-minded recursive descent compiler can generate remarkably good code. The DRI C compiler that comes with CP/M-68K seems to be much better than a lot of them out there, with about a 1.25-1 "HLL overhead," although with modern techniques, MUCH better is possible. Second, the simplicity of C comes in major part from the fact that it has no built-in functions, I/O primitives, etc., as many other languages do; all it has are facilities for dealing with the machine in the machine's own terms (pointers, chars, words, registers, etc.). Therefore, any I/O, etc., must be supplied from outside the language, from externally defined libraries. We put these two statements together, and a remarkable conclusion can be drawn: If a program written in C is inefficient, EITHER THE

THE PROGRAMMER'S SHOP™

helps save time, money and cut frustrations. Compare, evaluate, and find products.

SERVICES
- Programmer's Referral List
- Compare Products
- Help find a Publisher
- Evaluation Literature free
- BULLETIN BOARD - 7 PM to 7 AM 617-826-4086
- Dealer's Inquire
- Newsletter
- Rush Order
- Over 700 products

Free Literature - Compare Products

Evaluate products *Compare* competitors. *Learn about new alternatives.* One *free call* brings information on just about any programming need. Ask for any *"Packet"* or "Addon Packet": ☐ ADA, Modula ☐ "AI" ☐ BASIC ☐ "C" ☐ COBOL ☐ Editors ☐ FORTH ☐ FORTRAN ☐ PASCAL ☐ UNIX/PC or ☐ Debuggers, Linkers, etc.

RECENT DISCOVERIES

SMALL TALK for PCDOS - "Methods" has objects, windows, browser, inspector. PCDOS $239

"C" LANGUAGE

	OUR PRICE
MSDOS C86-8087, reliable	call
Instant C - Inter., fast, full	445
Lattice C - the standard	call
Microsoft C 3.0 - new	279
Williams - debugger, fast	call
Wizard C - full, fast	450
CPM80 - EcoPlus C - faster, SLR	275
BDS C - solid value	125
MACINTOSH: Hippo II	375
Megamax - optimizer, full	275
Consulair's MAC C, toolkit	395

Compare, evaluate, consider other Cs

BASIC

	RUNS ON	
BASCOM-86 - MicroSoft	8086	279
CB-86 - DRI	8086	419
Data Manager-full source	MSDOS	325
CADSAM-Full Btree, source	MSDOS	150
InfoREPORTER-multifile	PCDOS	115
Prof. Basic-Inter., debug	PCDOS	89
SCREEN SCULPTOR	PCDOS	115
TRUE BASIC - ANSI	PCDOS	125

Ask about ISAM, other addons for BASIC

SERVICE

ALL PRODUCTS - We carry 700 products for MSDOS, CP/M 86, CP/M 80, Macintosh and key products for other micros.

EDITORS Programming

	RUNS ON	OUR PRICE
BRIEF - Intuitive, flexible	PCDOS	195
C Screen with source	86/80	75
Epsilon - like EMACS	PCDOS	195
FINAL WORD-for manuals	86/80	215
PMATE - powerful	8086	185
VEDIT - full, fixed	*86/80*	119
XTC - multitasking	PCDOS	95

COBOL

Dig. Res-decent	MSDOS	525
Macintosh COBOL - Full	MAC	1850
MBP-Lev II, native, screen	MSDOS	885
Micro Focus Prof. - Full	PCDOS	call
Microsoft - Lev II, no royal	MSDOS	500
Ryan McFarland - portable	MSDOS	695

Ask about program generators.

LANGUAGE LIBRARIES

GRAPHICS: Halo for Turbo PASCAL	MSDOS	95
GRAPHMATIC - 3D, FTN, PAS	PCDOS	125
MultiHALO - fast, full-all lang.	PCDOS	220
File MGNT: BTrieve-all lang.	MSDOS	215
CIndex + - source, no royal	86/80	369
CTree - source, no royal	ALL	369
dBC ISAM by Lattice	8086	229
dB VISTA - "Network" Structure	*MSDOS*	465
PHACT - up under UNIX, addons	MSDOS	225
OTHER: C Utilities by Essential	MSDOS	129
Greenleaf - 200 +	MSDOS	159
SOFT Horizons - Blocks I	PCDOS	139
SCREEN: CURSES by Lattice	PCDOS	125
MetaWINDOW - icons, clip	PCDOS	139
PANEL - many lang. term	MSDOS	249
ProScreen - windows, source	PCDOS	415
Turbo V - Greenleaf C, fast	PCDOS	159
Windows for C	MSDOS	175

FORTRAN

	RUNS ON	OUR PRICE
MS FORTRAN-86 - Impr.	MSDOS	239
DR Fortran-86 - full '77	8086	249
PolyFORTRAN-XREF, Xtract	PCDOS	165

OTHER PRODUCTS

Advanced Trace 86 - Symbolic	PCDOS	149
Assembler & Tools - DRI	8086	159
Atron Debugger for Lattice	PCDOS	395
C Helper: DIFF, xref, more	86/80	135
CODESMITH-86 - debug	PCDOS	129
MacASM - full, fast, tools	MAC	115
MBP Cobol-86 - fast	8086	885
MicroProlog - improved	MSDOS	185
Micro: SubMATH - FORTRAN full	86/80	250
Microsoft MASM-86	MSDOS	125
Multilink - Miltitasking	PCDOS	265
PC FORTH - well liked	MSDOS	219
Pfinish - Profile by routine	MSDOS	345
PFIX-86 Debugger	MSDOS	169
PL1-86	8086	495
Polylibrarian - thorough	MSDOS	95
PolyMAKE	PCDOS	95
TRACE86 debugger ASM	MSDOS	115
ZAP Communications-VT100, TEK 4010, full xfer	PCDOS	65

Note: All prices subject to change without notice. Mention this ad. Some prices are specials. Ask about COD and **POs**. All formats available. UNIX is a trademark of Bell Labs.

Call for a catalog, literature, and *solid value*

800-421-8006

THE PROGRAMMER'S SHOP™

128-1 Rockland Street, Hanover, MA 02339

Visa Mass: 800-442-8070 or 617-826-7531 MasterCard 8517

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ALGORITHMS USED ARE POTTY OR THERE IS SOMETHING WRONG WITH THE RUNTIME LIBRARIES. Let us be charitable and assume that the authors would not knowingly use bad algorithms (although I have something to say about that, also) and go on to inspect the only thing left: the runtime libraries.

"In looking at the libraries supplied for three CP/M-80 C packages, and the CP/M-68K library, and seven C libraries for the (yuck!) 8086 crowd, I have observed a common element. EVERY ONE OF THESE PACKAGES TRIES TO MAKE ITS NATIVE OPERATING SYSTEM LOOK LIKE WHAT THE IMPLEMENTORS PERCEIVE AS UNIX! The farther the native operating system is from having the features of UNIX, the harder the implementors try to make it UNIX, with the result that every program carries the overhead of a UNIX imitation.

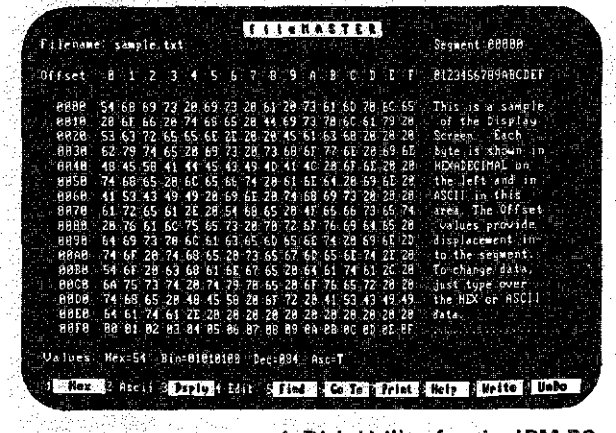
"Look at CP/M-68K, for example. If there was ever a system farther from the conceptions of UNIX than CP/M (for the personal computer market) I haven't heard of it. Yet the DRI C library will put ALL the code necessary for I/O redirection via the command line (a UNIX feature), buffered stream files (a UNIX feature), etc., ad nauseum, into every program I write. I've studied the code in the DRI library, and have come to the interesting conclusion that just to get a character on the screen using any of the library functions increases the instruction path length by about two orders of magnitude over using purely CP/M related calls, simply because of all the "we gotta look like UNIX" code. The famous "printf ("HELLO, world+n");"complies to 34 bytes, but when all that UNIX garbage is linked in to get an executable codefile, it has expanded to over 15,800 bytes, and for each character to print we go through all the garbage.

"I would dearly love to know who started this, to me completely idiotic, belief that "if it's C, it's gotta be UNIX." I'd be gentle with him, I promise; the poor person is obviously not all here. He probably spends a lot of time editing newsletters. I've heard that's good therapy . . .

"Anyway, it's this 'C = UNIX' attitude that is the cause of the inefficiency of a lot of programs, and not the low-to-medium level language C. My solution to the problem is to rewrite the library to be used with CP/M-68K, not UNIX. Funny, when I did this, programs shrank and sped up rather dramatically. Equally funny, the programs are portable to any CP/M-like system. You can't take them to UNIX, but I DON'T RUN UNIX! Which brings up the question of portability. I don't particularly care if programs I write for CP/M-68K aren't portable to UNIX or MVS/XA or what-have-you. Generally, different machines/systems/environments have different needs, so a really useful program in one environment is useless in another, or worse, actually detrimental. So it's enough for me to ensure that my programs are portable between systems sufficiently similar to have the same needs. I fail to understand the current trend of 'portability through mediocrity.'

"The next thing I want to say relates to CP/M-68K itself, and to the above charitable assumption about someone knowingly using bad algorithms. I have read that a great number of people are saying things like CP/M-68K is slow, the development tools are slow, etc. My first question is: Slow relative to what? And under what circumstances? I first observe that a lot of the newer machines are being designed with 5 1/4 inch drives; from my experience with mini-floppies, I'd say there's a major source of speed reduction, right there. My personal observations on my own 68010-based system (8 inch drives, and a meg of memory) is that CP/M-68K is, while not instantaneous, quite a bit faster than many other systems I've seen. The thing that blows CP/M-68K out of the water as a system is the abovementioned stupidity about algorithms. Here we have a processor with a gigantic linear address space, an incredibly efficient instruction set, and a very powerful systems language for generating good code. AND NONE OF THE PROGRAMMING TOOLS TAKE ADVANTAGE OF ANY OF IT! We are supplied with a line editor that won't hold 22K in a MEGABYTE of memory, assemblers and linkers that are totally disk to disk (as nearly as I can tell, even the symbol table is maintained on disk, f'r chrissakes!), therefore I/O bound, and SLOW, a file transfer program that only buffers 16K at a time, and a compiler whole libraries are optimized for ' compati-

fileMASTER THE DISK UTILITY



FINALLY!! **A Disk Utility for the IBM-PC that's fast and easy to use.**

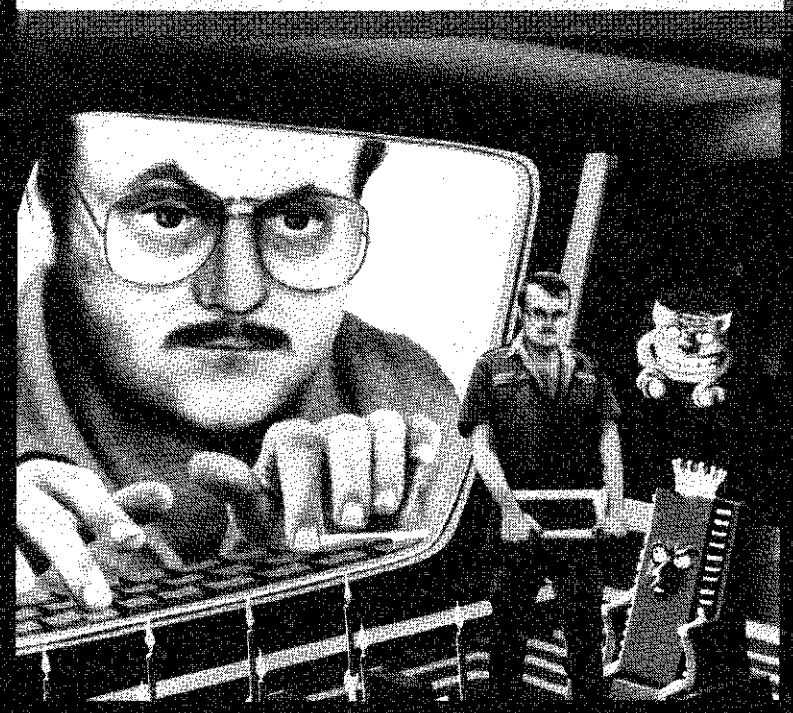
- FILE MODE
- DISPLAY
- PRINT
- MODIFY
- SECTOR MODE
- BROWSE
- SEARCH
- DOS 1.1 to 3.0

ONLY **$39.95** + $2.00 SHIPPING, CA + 6.5% TAX

J. L. SCHULLER & ASSOCIATES (818) 366-6934
14800 Rinaldi St., Suite 6, Mission Hills, CA 91345

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

??WHO IS THIS MAN??



HE IS—

A Brand-Name Show Stopper"
INFOWORLD

"The Dean of Computer Columnists"
SAN FRANCISCO CHRONICLE

"Challenging, articulate, and always opinionated…
followed by a
growing band of energized users…Jerry makes us think about how the wonderful
machines will change our lives and shape our future."
—*POPULAR COMPUTING*

"His 'Users Column' has been one of *Byte* magazine's most
popular columns for years…he states his opinions fully, without compromise,
and in a lively colloquial style…no one could copy his style."
—*BYTE*

HE'S ALL THAT AND MORE HE'S AMERICA'S #1 USER!

JERRY POURNELLE

AND HE'S BACK, MORE POPULAR THAN EVER, IN

ADVENTURES IN MICROLAND

For everyone who read *The Users Guide to Small Computers*
and now just has to have more of the wit and wisdom
of America's Number One User, here it is!

July • 55980-0 • 416 pp. • $9.95

BAEN BOOKS

Distributed by Simon & Schuster Mass Merchandise Sales Company
1230 Avenue of the Americas • New York, N.Y. 10020

PRODUCED 2024 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

bility' (Lord, I'm beginning to hate that word!) with a DIFFERENT operating system!

"It is fairly clear that CP/M-68K was an afterthought on the part of DRI, whose efforts are concentrated on the 8086 and Itty-Bitty-Monstrosities compatibility (there's that word again!), but there is no excuse for this kind of nonsense (when I use my virtual disk as the work drive, though, things speed up considerably, the limiting factor being the size of the programs being loaded and the programs being compiled). This is what I meant by using stupid algorithms; the system has calls for determining available memory, so why not use them and keep as much as possible in core? Why not autoload the passes of the C compiler, using the built-in program load function, instead of having to use the primitive batch facility? Why not . . . ? You get the picture. The funny thing is that ALL of these features are available on CP/M-80!"

Daniel R. Lunsford
4735 Marconi Ave.
Carmichael, CA 95608

(We are printing Dan's address with his permission, so that you may write to him directly if you wish.) By the way, when we asked Dan how one could achieve MUCH better performance than 1.25-1, he asserted that modern compiler techniques permitted 1- 1! That agrees with the book cited earlier by Bill H. Hmmm . . . .

AN IMPORTANT SUBJECT?

We believe that the question of the efficiency of a high level language, especially one which is supposedly suitable for use in writing operating systems and/or other high level languages, is an important one. We also believe that if any high level language can produce, as a finished product, something which is only between 0% and 25% slower than the same product as developed by a skilled, experienced assembly-language programmer THEN IT IS A GOOD IDEA TO USE THAT HLL! (Barring the occasional real-time application which cannot tolerate even a 25% performance degradation.)

Until now, we had not considered that the question of the HLL overhead of a language might be different than that observed in finished products such as WordStar 2000 or (ugh!) DRI's Personal BASIC. Daniel L. has just produced a strong argument that the horrendously slow performance of observed finished products written in C is not due to the C language itself. This immediately raises the question, "If finished products produced using C invariably have abysmal performance, as in benchmarking with an almanac, why should the end-user give a fig whether or not the language itself is at fault? Is not the abysmal performance itself to be avoided?" We believe that Pournelle, in his *BYTE* "USER'S COLUMN," once observed that he could not evaluate programming languages, just IMPLEMENTATIONS of programming languages!

If Daniel L.'s argument has merit, is it not appropriate for the writers of finished products to get rid of those high-overhead I/O routines? And for *BUYERS* of finished products to avoid finished products written in C until that is accomplished?

Our first reaction after reading Dan's letter was that the producers of those finished products would not be so stupid as to kill the performance of their products with highly inefficient I/O. On reflection, we have changed our mind. The producers, essentially 100% of them, COULD well be so stupid. Remember, the folks working with C are indeed disciples of the great god TRANSPORTABILITY. And, after all, those finished packages were probably developed in a UNIX environment! You will remember that UNIX machines are great for developing software — and anybody who has followed UNIX knows that all that software being developed must be sold elsewhere because it sure as hell is not being sold to the UNIX marketplace! And remember, a main feature of UNIX is that all of the utilities — such as I/O routines — are already written, so that the software developer does not have to constantly reinvent the wheel. Given that philosophy, the problems cited by Dan automatically follow.

On the other hand, we believe that Dan has only part of the picture. For example, the horrid "performance" of the AT&T 3B2/300's floating point package has been thoroughly documented and confirmed. It does not seem possible to blame that on I/O routines since to the best of our knowledge floating point does not use I/O!

Another point: When someone asserts that a compiler "produces code just as good as assembler," just exactly what does that mean? Does it mean that the compiler produces code which is just as efficient as that produced by a graduate student who has just been handed an Intel 8086 programming manual and who has never written assembly code before in his life? Or is the claim being made that the compiler produces code as good as the guy who did most of the assembly coding of Lotus' 1-2-3 (NOT Mitch Kapor, as it turns out)?

If the compiler generates code modules which set up parameters on a stack, call a subroutine, and leave the results on the same stack, is the efficiency of the compiler judged against assembly code WHICH IS CONSTRAINED TO USE THE SAME TECHNIQUES or against assembly code written using whatever technique works best, in the opinion of the assembler?

If we can take an inexperienced graduate student and make him use the same ground rules as those used by the compiler then we can most certainly produce code whose high-level overhead is vanishingly small.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Instant-C™: The Best Value In C Programming Tools

The edit-compile-link-test-debug cycle that takes tens of minutes with compilers and linkers is only seconds with the *Instant-C* interpreter. Yet it runs your programs **50 to 500 times faster** than conventional C interpreters! You get the best of both compilers **and** interpreters. Only *Instant-C* is a complete, integrated environment for creating, testing, and running your programs.

Instant-C gives you **all** of these proven capabilities in one tightly integrated package:

interpreter—*Instant-C* runs your programs faster than some compilers; has direct execution; full K&R

compiler—*Instant-C* can make stand-alone programs

full-screen language editor—shows syntax errors with cursor set to trouble spot

C source debugger—single-step, breakpoints, stack trace, more

run-time checker—validates pointer refs, array bounds, more

C source formatter—save editing time, find logical flaws

standard library with source—for best portability

linker—work with multiple source modules

Lint—extensive compile-time validation

The cheapest available examples of these tools would cost $800 (and they don't even work together). You could spend close to $3000 to get the best product of each kind, but you'd have ten times the complexity, filling megabytes of disk. *Instant-C* is faster: it performs these functions automatically. *Instant-C* is far more than the sum of its parts.

Instant-C is all of these capabilities in one package, fits on a single floppy disk, is full K&R, works on IBM PC's, compatibles, and others under DOS or CP/M-86. It costs only $495.

Instant-C is the best value in C programming tools. Guaranteed, or your money back for any reason in first 31 days.

Rational Systems, Inc.

(617) 653-6194
P.O. Box 480
Natick, MA 01760

Instant-C is a trademark of Rational Systems, Inc

Moving?

Please send us both your old and new addresses, with Zip Codes. If possible, include the address label from a recent issue of *Programmer's Journal*. Allow three to four weeks for processing.

Programmer's Journal
P.O. Box 3000
Dept. EE
Denville, NJ 07834

YOU NEED A GOOD LIBRARY

POWER PACKS™

COMPLETE SOURCES NO ROYALTIES

COMPREHENSIVE C Power Packs include over 1000 functions which provide an integrated environment for developing your applications efficiently. "*This is a beautifully documented, incredibly comprehensive set of C Function Libraries.*"
– Dr. Dobb's Journal, July 1984

USEFUL "...can be used as an excellent learning tool for beginning C Programmers..."
– PC User's Group of Colorado, Jan. 1985

FLEXIBLE Most Compilers and all Memory Models supported.

RECOMMENDED "I have no hesitation in recommending it to any programmer interested in producing more applications code, using more of the PC capabilities, in much less time." – Microsystems, Oct. 1984

- **PACK 1: Building Blocks I** $149
  DOS, Keyboard, File, Printer, Video, Async
- **PACK 2: Database** $399
  B-Tree, Virtual Memory, Lists, Variable Records
- **PACK 3: Communications** $149
  Smartmodem™, Xon/Xoff, X-Modem, Modem-7
- **PACK 4: Building Blocks II** $149
  Dates, Textwindows, Menus, Data Compression, Graphics
- **PACK 5: Mathematics I** $99
  Log, Trig, Random, Std Deviation
- **PACK 6: Utilities I (EXE files)** $99
  Arc, Diff, Replace, Scan, Wipe

Master Card/Visa, $7 Shipping, Mass. Sales Tax 5%

ASK FOR FREE DEMO DISKETTE

NOVUM ORGANUM INC.

SOFTWARE HORIZONS INC.

165 Bedford St., Burlington, MA 01803
(617) 273-4711

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

We believe that this question IS important. We know of a company much larger than Digital Acoustics, but which is small enough that upper management cares about such issues, that is performing an in-depth study of the question of HLL efficiency right now.

MAYBE NOT IMPORTANT!

The MIS departments of large companies are frothing at the mouth to regain control of all those personal computers out there. Every personal computer which is not controlled by the MIS department diminishes the importance of the department and its honcho — horrors! We were therefore interested to read, in *DATAMATION*, about how individual users select PCs vs. how the MIS departments do the same selection. Individuals judge 'PERFORMANCE' to be the third most important factor, while 'PERFORMANCE' does not even appear on the MIS list of factors to be judged! I YI YI!

The underlying theme of this rag, of course, is just how much performance an individual can get out of hardware which that individual pays for out of his or her own pocket and which the individual expects to use himself or herself. And it certainly does not make sense — to us — to select the best available and affordable CPU (the 680X0 series) and then cripple it with unnecessary HLL overhead.

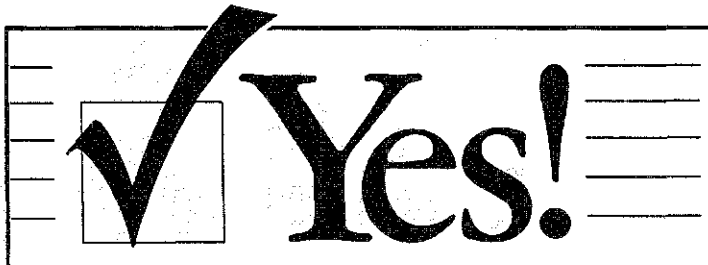
Perhaps it makes sense to cripple the performance of a company computer for the sake of convenience, and it is undeniable that HLLS are convenient. But (please excuse the redundancy) this rag is NOT about company computers! (So will certain readers stop telling us how good virtual memory is when run on big, expensive company computers? Please?)

FNE GOOFS AGAIN! (YAWN):

Until recently, we had received about three letters and one personal visit (by the owner of a DIMENSION $5000 68000 machine) discussing CP/M-68K, and all of those early contacts reported that CP/M-68K was slow and took up nearly 100K memory space. We have subsequently received about three letters, including Daniel L.'s, which assert that CP/M-68K is NOT slow and that it consumes about 22K memory space. Obviously, there is a difference of opinion here, and we were premature in forming our opinion. We apologize to you readers for our carelessness and to DRI for, perhaps, incorrectly maligning their product.

Let us make it clear that we are not switching sides, but rather that we now take a neutral stance. We would like to hear more from you readers on this subject, and suggest the following guidelines:

"FAST" and "SLOW" are relative concepts. If you are going to discuss the speed of CP/M-68K, please do so in comparision to a contemporary, competing operating system (preferably MS-DOS) using comparable hardware. (One reader wrote that his CP/M-68K was reasonably fast using a one-megabyte RAM disk!) □



Please enter my new subscription to

programmer's journal

Bi-Monthly Subscription Rate

	U.S.	Canada (U.S. Funds only)	Foreign
1 year	☐ $24	☐ $30	☐ $40
2 years	☐ $44	☐ $55	☐ $75
3 years	☐ $64	☐ $80	☐ $110

Mail to: P.O. Box 3000 Department EE
Denville, NJ 07834

☐ New Subscription
☐ Renewal

☐ Check enclosed
☐ Bill me
☐ Charge my Bankcard: ☐ VISA ☐ MC
Card No. _____
Expiration date _____
Signature _____

Name _____
Company _____
Address _____
City _____ State _____ Zip _____
Telephone _____

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C PROGRAMMING BOOKS

survey by Frank D. Greco

Frank Greco recently tangled with some twenty-four books on the C language, with, as you'll see, mixed results. In this issue of Programmer's Journal *we present his evaluation of the first twelve of the lot. Next month we'll publish the second half of this comprehensive critical review.*

Since the popularity of both the C Language and the increasingly ubiquitous UNIX operating system has been on the rise recently, a sudden plethora of books has appeared. This recent onslaught of C and UNIX books has been both a boon and a bane to program developers and students of the world. As with other recent computer-related books, there are the usual proportions of low and high quality UNIX and C language works. Many of the low quality texts out there are likely due to misperceptions/misunderstandings that some authors have about their subject. With this misguided information about the C language, the general programming populace grows increasingly confused.

For example, in a recent major publication a reporter says, "The language [C] is not yet a popular business language [!] because of its problems with disk-file and high-precision arithmetic support." See what I mean? Apparently this reporter for a widely read magazine thinks that C is, or should be, a variant of COBOL. C professes no hereditary linkage with COBOL; this sleek language doesn't even acknowledge a casual relationship with that overweight, dirty old language. Also, I fail to see the problems that C has with disk-files or arithmetic support. Perhaps the reporter wants file access methods (ISAM, VSAM, . . .etc.) and BCD arithmetic built-in to the language. C is not a panacea (no language is); it merely fills a certain niche in the programming language hierarchy.

Because of frequent misinformation about the C language and the UNIX OS in the popular computing "literature," as exemplified by the example above, one should separate the wheat from the chaff. Lately, wheat has been a somewhat rare commodity. I would like to share with you some personal comments on some of these books. Let's look at some books on the C programming language first.

1. *The C Programming Language* — Kernighan/Ritchie — Prentice-Hall

A definition of the "C" programming language lies between the covers of this surprisingly small book. The gentlemen, both of whom are currently employed by AT&T Bell Laboratories at Murray Hill, New Jersey, have written a classic, definitive, concise, and occasionally cryptic work. Dennis Ritchie is the designer of the C language (modeled after the B language created by UNIX inventor Ken Thompson), and Brian Kernighan is the computer scientist who coined the name "UNIX" (a ruthless pun on UNIX's unwieldy ancestor MULTICS) for the extremely flexible operating system that we all know and love/hate. Their book, commonly referred to as "K&R," is a necessity for serious C programming work. If you are a newcomer to structured languages (Pascal, Modula II, Ada), this book should be used only as a reference, NOT as a tutorial.However, once you have about 6 months experience with the C language and start seriously to read portions of this book, you will be amazed at what you will learn!

One small problem of timeliness, however, remains with this book. The K&R book was published in 1978 and coincides with UNIX Version 7 (hence many vendors call their C compilers V7 compatible). Since that time there have been substantial enhancements to the language, two updated versions of the language ("C++" and "Concurrent C"), and a recent standardization effort by an ANSI committee.

The enhancements to the standard language that are not in the book include passing a COPY of a structure to a function, copying an entire structure (assignment of structures), functions that return a non-value (void), and enumerated types. Another little known enhancement deals with the C language under a UNIX environment only: the passing of three variables to main(). These variables are conventionally called argc (argument count — an int), argv (list of arguments — array of pointers to char) and arge (list of environment variables — array of pointers to char); i.e.,

```
void main(argc, argv, arge)
int argc;       /* number of arguments */
char *argv[];   /* list of arguments */
char *arge[];   /* list of environment variables */
```

Note that the names "argc," "argv," and "arge" are merely conventional; you can use whatever names you want.

In regard to the struct enhancements, K&R say that only a pointer to a struct could be passed to a function; you could not pass a copy of a struct to a function. And, structs could not be copied via an assignment expression. K&R, knowing that these deficiencies should be removed in future releases of

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

the language, had the foresight to at least mention (page 121) "... restrictions (on structs) will be removed in forthcoming versions. . . ."

Remember, however, that although assignments on structs are allowed, testing for logical equality is still not allowed. For example,

```
struct {
     int x;
     int y;
     char c;
     } str1 = { 0, 1, 'C'),  str2;

str2 = str1;          /* This is allowed */

if (str1 == str2)     /* This is still NOT allowed */
     ;
     ;          /* double equals is logical equality */
     ;
```

The other enhancement, the introduction of enumerated types, was developed to quell the aching groans of those Pascal programmers out there. So now, new types, like COLOR, LIGHT, and ALIEN_TYPE (for you videogame hacks out there), each with a clearly defined set of possible values, can be added to the language. Some of you are saying, "What's the big deal? #DEFINEs can be used to simulate enumerated types." That's true; however, enumerated types are types that are added to C's knowledge domain, and as such, C can type-check these new types reasonably well. Macros (i.e., #DEFINEs are only textual replacements and are the domain of the C preprocessor, not of the compiler proper.) Unfortunately, not all C compilers for the PC marketplace support enumerated types.

As far as C standardization is concerned, I have been following, with much amusement, the recent debate on the ANSI effort in the ACM's SIGPLAN journal (Association of Computing Machinery — Special Interest Group on Programming LANguages). Some academics say that K&R is standardization enough, while the commercialists want file access methods (like ISAM) in the standard library. Well, we all know about standards don't we? After the ANSI effort, I'm sure we will see advertisements like "New, ANSI Standard C Compiler WITH ENHANCEMENTS!"

C++, as mentioned in *PJ*, Vol. 2, No. 2, is one of the newer versions of the C language. Created by AT&T Bell Laboratories' Bjarne Stroustrup, the biggest additions to the C language that C++ offers are classes (or modules). Simula, Ada, and

THE HAMMER

Software Tools in C

Designed for the programmer with the user in mind.

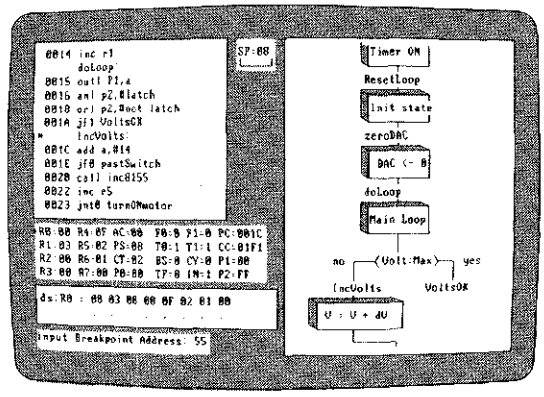
More than just BIOS/DOS access. , THE HAMMER ALSO provides screen attribute control, Smartmodem control, UNIX-like pattern matching, 123-like command menus, date & time processing routines, easy data entry & verification of dates, numbers, and strings, AND MORE. Super data entry routines give the programmer an easy to use, solid interface with the user.
Requires: IBM-PC/XT, DOS2, CI-C86 Compiler
Other C compilers supported soon.

Routines tried & true. $110 with source and manual.
For ordering or inquiring, call or write now:

O E S SYSTEMS
1906 Brushcliff Road
Pittsburgh, PA 15221
412/243-7365

Looking for the right tool for the job?
GET THE HAMMER

Simulator/Debugger for IBM-PC for 8048 - 8051 - 7000 µCs



Execute and debug code for popular single chip microcomputers on your IBM PC. Dynamic display with windows for source code, control flow, registers, flags, memory, commands, and more! Set breakpoints, traps, etc. Cross-assembler & EPROM programmers, too.

	8048	8051	7000	Z8	
X-Assembler	✓	✓	✓	✓	$195
EPROM Prog	✓	✓	✓	✓	$245
Simulator	✓	✓	✓		$395

Demo disk and manual for simulator only $39.50

Cybernetic Micro Systems
Box 3000 • San Gregorio, California 94074 U.S.A.
(415) 726-3000 • Telex 171-135 Attn: Cybernetics

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Modula possess these concepts. So far, C++ (formally) exists only within the walls of AT&T Bell Laboratories. Rich Mascitti, who named this newest version of C, presumably gave it the name "C++" because when you increment a variable (the "++" operator), you denote a succeeding entity. It's just a guess, but I am a bit familiar with the offbeat humor at AT&T Bell Labs.

The other new version of the C language is called "Concurrent C" and was created by AT&T Bell Laboratories' Concurrency maven, Narain Gehani. A paper on Concurrent C was delivered at the January 1985 Usenix conference in downtown Dallas. The biggest additions that Concurrent C makes to the "ordinary" version of C are a set of constructs that handle concurrent programming. Concurrent programming is the study of programming tools or techniques that deal with parallel processes. This is especially important in systems that link many processors together. Like the Ada language, Concurrent C is based on a synchronous message passing scheme (the rendezvous model). Similar to C++, Concurrent C lives only within the walls of Bell Laboratories.

All in all, even though it isn't 100% timely, the Kernighan and Ritchie classic should be in every professional C programmer's library.

2. *The C Primer* — Hancock/Krieger — McGraw-Hill

The C Primer is exactly what it proports to be, a beginner's look at the C language. The authors draw many comparisons with the BASIC programming language. This is good for those programmers out there who know and think only in BASIC.

However, it is highly arguable whether or not beginning programmers should learn the C language. Remember that the C programming language is NOT a very high level language. The author of the C language, Dennis Ritchie, says that it is a "relatively" low level language. It is commonly referred to as a generic (that's read "portable") assembler language; some other programmers call C a structured assembler. New programmers should really learn higher level languages to get accustomed to high level looping constructs, procedure/function calls, abstract types, etc.

Another problem with the authors' approach is that programmers who already know BASIC and now want to learn the C language should start with a "clean slate" and not try to seek similarities between the two computer languages. For example, try explaining the methods of string handling in both languages to the beginning programmer. BASIC treats strings as high level entities, but C, formally speaking, does not have the concept of a string. A series of contiguous bytes that is terminated with a binary zero (ASCII NUL) is C's loose concept of a string. BASIC has all these built-in functions to handle substrings and string lengths, but C has no string handling. All string handling in a C program is performed by functions that are external to the language.C is not related to BASIC and should not be taught that way. Comparing BASIC to C is like comparing apples to pi mesons.

Fortunately, these two gentlemen are excellent authors and convey the material as well as possible to those just starting out. I would not recommend this book to a fairly experienced programmer as it leaves these readers thirsting for more, especially in the discussion of pointers. Assembler and Pascal jocks should stay away from this particular book. However, the book is obviously directed at the neophyte programmer, and with such an audience, it passes as a better than average book on C.

3. *Programming in C for the Microcomputer User* — Traister — Prentice Hall

The only item of interest in this book is the prologue (four pages) on the origins of the C language and the UNIX operating system. The rest of the book has some SERIOUS errors.

The first looping construct that the author explains is the GOTO. Yes, you did hear me correctly. 'Nuff said on that one.

Another strange statement Traister makes is ". . . a counting sequence would normally be written as: z = ++1." What the heck does that mean? "z = ++1" is syntactically invalid (you cannot increment an "rvalue")!

In every example in his book, the author uses some strange and confusing stylistic constructs. Traister does not like to indent logical blocks of code consistently. Didn't we all learn that in Programming 101? It HAS been proven that simple indentation will decrease programming errors dramatically. Also, isn't a good programmer supposed to choose variable names that are mnemonic? On page 104, Traister uses the following variables: i, x, ii, xx, y, z, and of course, zz. You are expected to remember that z represents xx squared.

On page 46, the author uses a dangerous construct that is not guaranteed to work the same on any two C compilers; that is,

```
int x;
x = ++x;  /* this is NON-PORTABLE */
```

This particular construct proves that Mr. Traister is very unfamiliar with the C language. If he knew the language, he would realize that he is relying on order of evaluation here and should not use constructs like this. Although precedence is clearly defined in C, order of evaluation is NOT defined at all (except for left-to-right evaluation of &&, II and ,). K&R

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

specify this on pages 48 through 50 of their classic. A possible corrected version of the above would be the following guaranteed-portable program segment:

```
int x, y;
y = ++x;
x = y;
```

Additionally, I could not even find a simple table of operator precedence and associativity in this book. How could a student possibly learn the language without one? I personally have copied page 49 of K&R and taped it to the wall near my machine for quick reference.

Traister's knowledge of C's string handling is sadly lacking. On page 77 he says, "When dealing with [stringf] variables in C . . . you must [!] specify the variable type before you perform any operations on them." If Mr. Traister had read the C Bible (K&R), he would have realized that there is only one default type in the C language, and that is "int." It is important to raise this point for students of the language, not so that they'll use the default (a poor practice to say the least), but for debugging purposes. For example, a common bug for beginning C programmers is failing to declare a function that returns a non- integer. C is loosely typed; it is not as strongly-typed as Pascal or Modula.

Another blunder shows up on page 79: " . . . C programming language has a number of functions . . . specifically for comparing character strings and even merging them." This is wrong! C has NO intrinsic string handling functions. String handling is performed through the use of standard library functions, which are typically (but not necessarily) sold along with a compiler. By definition, all C functions are of storage class external; that is, you cannot define (i.e., create) a function inside a function.

Obvious mistakes exist in practically every example in the string handling chapter. For example (page 81),

```
char x;
x = "hello"; /* This is WRONG ! */
puts(x);     /* note that 'puts' assumes its
                argument is a char pointer */
```

This is incredibly incorrect! You cannot perform this operation consistently in C. Mr. Traister believes that he is assigning the string "hello" to a variable *a la* BASIC. What the program segment above is doing, however, is trying to assign a pointer to start of the bytes 'h', 'e', 'l', 'l', 'o', '\0' to a character variable. The resultant number that is placed into x is surely unreliable since pointers are typically larger than 127, which is the largest bit pattern that can fit into a C char variable (unsigned chars can handle a full byte).

I tried duplicating Traister's very small program, and, it worked. It worked because the address of where "hello" was stored was less than 127. I then tried the segment above in a much longer program, insuring that the value of the pointer was greater than 127. I did this by creating at least 128 bytes of static data before defining "hello" (strings are typically stored in static data space). When I ran my test program on my PC/XT, all I saw was a series of control characters. So this subtle but pernicious bug will not surface unless the value of the pointer is greater than the capacity of a char type variable (decimal 127). Even though most C compilers will allow this mismatched type expression due to C's loose type checking, they should at least warn the programmer. Computer Innovations' C86 did not complain at all, but my C compiler running under BSD 4.2 UNIX at least warned me; i.e.,

```
        void main()
        {
                static char string[] = "This is some data that
will be stored in the static data area of a C program.  As lon
as I have at least 128 characters in this array, I will insure
that  the next string starts after address 127." ;

                char x;

                x  =  "hello";  /* "hello" will start in memory
                                   after above string */
                puts(x);

        /* Your output should be nothing but garbage */
```

The entire chapter on strings is completely and embarrassingly incorrect, the organization is extremely poor, programming style is awkward, and there are no illustrations to accompany the lackluster text. Apparently, Traister never had anyone proofread his manuscript before publication. This book is to be avoided.

4. *C Programming Guide* — Purdum — Que

Jack Purdum's book comes highly recommended. It is well organized and extremely clear to the C language student. I even like the "easy on the eyes" font used in the typesetting process. Highlights of this book include the excellent examples of structures utilizing crt display control, the chapter on "a Closer Look at File I/O," the discussion on pointers, and a chapter on "Common Mistakes and Debugging." The "Mistakes" chapter is well worth reading several times over as Purdum obviously has gained some valuable experience teaching the language.

There are only a few places where Purdum slips up. On page 66 he says, "Bear the distinction [!] in mind: Functions receive copies of all variables except numeric and character arrays." Well, this is slightly misleading. You cannot pass an array to a function in C; you pass a pointer to the array. Typically, the pointer passed is the name of the array by itself, which is the address of the zeroth element. C passes parameters ONLY by value. The called function "sees" only VALUES sent from the caller. If you want to purposely cause a side effect and alter the value of an input argument to a function, you can pass the

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

VALUE of a pointer to the variable. However, the function had better receive the incoming parameter into a pointer type. Neglecting to declare the receiving variable as a pointer is a very common (semantic) error when simulating passing by reference (*a la* BASIC or FORTRAN).

On page 207, Purdum says, "A pointer contains garbage until it is initialized." Although this is a safe axiom to adhere to, it really is somewhat incorrect. If a pointer variable is of an automatic storage class, then the author is correct. Because automatic variables are created on the stack, their values are initially undefined (or garbage).If the pointer variable is of the static or external storage class, then the pointer contains binary zero ('\0'). As a matter of fact, all externals or static variables are, by default, initialized to numeric zero at compile time. In C, when a pointer variable is zero, the programmer is assured that it points to nowhere (as opposed to pointing to memory location 0).

While these complaints are really about trivialities, my only substantial criticism is of the author's use of practical examples. I thought that most of his examples were contrived and not truly exemplary. However, even though it is spotty with examples, I would strongly suggest this book to the C language newcomer.

5. *The C Puzzle Book* — Feuer — Prentice Hall

Warning: The Programmer General warns you that this book may be harmful to your health, . . . at least, if it is the first C book that you read. Alan Feuer's (pronounced "Foy-er") intriguing little book is not to be read until you have a lot of experience with the C language. This book is a collection of program samples and corresponding "what is the output?" types of questions. The answer to each puzzle is given on the page following the exercise. Examples involving side-effects, structures, pointers, precedence vs. order of evaluation, and other topics are covered. Beginners should not use this book as a first introduction to this Spartan language; give yourself at least six months programming in the C language before tackling this Fearsome Feuer-some.However, if you can do all the examples in the book, you indeed have a mastery of the C language.

6. *C Notes* — Zahn — Yourdon Press

This is a very early book on the C language. It was published in 1979 right after K&R. While most of the information in this little book (96 pages) is correct, the organization is sketchy. I disagree with Zahn on one point — when he says "that there is no concept of a pointer constant only pointer variable names [?], pointer function calls, and parenthesized pointer expressions." First of all, the name of any array is a pointer constant. It points to the zeroth element. It is a constant because you cannot change it. Also, the address-of operator (&) renders a pointer constant. And the value of a string constant (e.g., "hello" — note the double quotes) is a pointer constant; it is actually a pointer-to-a-char constant. For example,

```
char line[80], ch;
        :
line = 128;  /* WRONG! "line" is a constant not a variable */
        :
&ch = 256;   /* WRONG! "&ch"  is also a constant.*/
        :
"hello" = 1700; /* WRONG! You cannot change where
                        the string 'lives' */
```

I also took umbrage at Zahn's description of C as "an implementable ALGOL 68." I hope this was tongue-in-cheek; it certainly was a subjective statement about ALGOL 68 rather than C.

The author assumes that the reader is experienced and already versed in another structured language. For example, recursion is brought up as early as page 14, and complex data declarations are formally described on page 29. So, while most of *C Notes* is correct, I really would not recommend it; there are better books on the topic.

ASCII Convert

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

7. *Learning to Program in C* — Plum — Plum Hall Publ..

This is a so-so work. The author does not assume any prior knowledge of structured programming: "Presupposing only an acquaintance with computers . . . ." There are some inconsistencies on structures. Plum says " . . . in standard C one cannot pass a structure as an argument to a function, but one may give its address as an argument." On the next page Plum says, "Furthermore, a structure may be passed as an argument to a function . . . ." Looks like one page was written before the changes to the C language (regarding structs) and one page was written after the updates. One highlight of the book is its pointing out that the "dot" (structure member) and "arrow" (structure pointer) operators can each be written in terms of the other.

For example, if we have the following declaration/definition:

```
struct EXAMPLE{
     int field;
     char name[100];
} s, *sptr;        /* s is instance of EXAMPLE */
             /*  sptr is instance of ptr to struct EXAMPLE */
```

Now,

```
sptr = &s;      /* initialize sptr to point to s */

sptr->field;    /* is same as (*sptr).field */
s.field;        /* is same as (&s)->field */
```

Note that the last two C statements above are syntactically correct; however, they constitute semantic nonsense. That is, although most constant expressions are legal C statements by themselves, they don't buy you anything useful. For example, "100;" is also a valid C statement, but it is of no practical use.

Other valuable parts of this book are the occasions when Plum focuses on stylistic concerns: when one construct is recommended over another for readability or portability. This is commendable. C is so flexible that it takes a lot of discipline to write readable and portable code.

The chapter devoted to software development is contrived. This topic alone can cover volumes, but Plum attempts to concoct an example that exemplifies software engineering problems in 29 pages. Some good ideas are presented in this chapter, but it leaves the reader (not unexpectedly) hanging.

Like the Zahn book, this one is, thankfully, correct; however, there are better.

8. *Comparing and Assessing Programming Languages Ada, C, and Pascal* — Feuer/Gehani — Prentice-Hall

This book contains a collection of various papers that evaluate and criticize these modern languages. Actually, this book contains a list of "what I think of Ada, C, or Pascal" type articles. Although some interesting papers are included in the book, it really is for the Computer Science major studying features of these languages. This is not a book for the hobbyist or casual user of programming languages.

For those who are interested, some of the works included in this book (edited by two C language mavens) are:

- "A Comparison of the Programming Languages C and Pascal" by Feuer and Gehani (written when they both worked at Bell Labs)
- "Pascal versus C: A Subjective Comparison" by Prabhaker Mateti (from the University of Melbourne)
- "Why Pascal Is Not My Favorite Programming Language" by Brian W. Kernighan
- "Type Syntax in the Language C: An Object Lesson in Syntactic Innovation" by Bruce Anderson

I must stress that this book does not teach the reader how to program in the languages mentioned; it just offers unabashedly subjective analysis.

9. *Programming in C* — Kochan — Hayden

Stephen Kochan, currently of AT&T Bell Laboratories, has written a well thought-out book on the C language. Like many other fine instructors, Mr. Kochan believes in the programming-by-example learning philosophy. It is difficult to open to a page in this book and not find an example of some sort. About 350 pages worth of information comes in this package.

Although the cover says that this book is for the novice and experienced programmer alike, I believe that advanced programmers might be a bit disappointed. The more interesting material for the advanced crowd lies in the exercises at the end of each chapter. These exercises do not have their solutions printed for the edification of those intrigued enough to attempt them.

What I really do not like about the book are the diagrams that illustrate pointers and/or structures. Instead of using ASCII characters to create boxes and arrows, why not use a graphic artist (or a graphics oriented computer)? Concepts would be a lot clearer.

Unfortunately, I spotted a few errors in the book. (As a C language consultant/instructor for several years, familiar with the usual student miscues, I tend to see C language errors in reverse video!) The first error is that Mr. Kochan confuses the issue of Definition vs. Declaration in the language

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

(page 323): "The definition of a . . . structure, union, enumerated data type, or typedef does not cause any storage to be reserved by the compiler . . . " This statement is wrong. The DEFINITION of any variable does indeed cause storage to be allocated. The DECLARATION of a variable does not allocate storage; declarations communicate information about data that is defined elsewhere. What other languages call "declaration" is called "definition" in the C language. For example,

```
struct PC
{    char model[10];
     int ram_size;
     char monitor_type;
};

/* The above is a data DECLARATION, not a definition */
/* Nothing is being allocated  */

struct PC
{    char model[10];
     int ram_size;
     char monitor_type;
}ibm;

/* This a data DECLARATION followed by a DEFINITION */
/* ibm is an INSTANCE of struct PC */
/* ibm is the variable being allocated */
```

Another slight error is on page 217, "Pointers and Arrays": "One of the most common uses of pointers in C is as pointers to arrays." Well, formally speaking, Kochan is in error when he says "pointers to arrays." He really means " . . . pointers to elements of an array." Pointers to arrays is another concept entirely. For example,

```
char *cptr;      /* cptr is a pointer to a character */
char list[10];   /* list is an array of characters   */
char  (*ap)[10]; /* ap is a pointer to an array of chars*/
        :
        :
cptr = &list[0];  /* initialize cptr to beginning of list */
  ap = &list[0];  /* initialize   ap to beginning of list */
        :
    cptr++;         /* cptr now points to next char */
    ap++;    /* ap now points to next array of 10 chars */
```

This error can be forgiven as it appears that Kochan knew what he wanted to say, but did not say it right.

For the programmer who needs tons of examples, Kochan's *Programming in C* is highly recommended.

10. *Programmer's Library* — Purdum/Leslie/Stegemoller — Que

This book is for the somewhat experienced C programmer. Essentially, the C Programmer's Library contains a large (contrived) example in which you can read the code and "find out what's going on." For samples of C code, *Dr. Dobbs'* or *Programmer's Journal* magazines always contain documented C source listings for different applications and are a much better buy.

11. *A Book on C: An Introduction to Programming in C* — Kelley/Pohl — Benjamin Cummings Publ.

This book ranks with Kochan's *Programming in C* and Purdum's *C Programming Guide* as one of the best books on the C programming language (I consider it the best). Unlike as in some of the other atrocious books on C, the authors split the book into beginning C and advanced C units. The obligatory introduction to C is extremely well done, but the advanced portion of the book is excellent. There are plentiful examples all through this book. As with the Kochan book (*Programming in C*), it is difficult to open the book on a page that does NOT have an example. Advanced topics such as data compression, numerical analysis, list processing, and stack manipulation are covered with source code examples for the reader to follow. Enumerated types, a new addition to most micro versions of the C language (enumerated types were formally added to the language right after K&R was published in 1978), is covered as well.

The only serious mistake that the authors make is the usual confusion over DEFINITION vs. DECLARATION. See my comments in the review of Kochan's *Programming in C*. I wonder why so many authors confuse this issue. K&R make the difference very clear on page 76 of their classic: "A declaration

NEW!

Advanced Trace86™

Symbolic Debugger & Assembler Combo

- Full-screen trace with single stepping; Even backstepping!
- Write & Edit COM & EXE programs
- Conditional breakpoints (programmable)
- Switch between trace and output screen; Or set up two monitors
- 8087, 80186, 80286, 80287 support
- Write labels & comments on code
- Polish hex/decimal calculator
- and more . . . Priced at $175.00

To order or request more information contact:

Morgan Computing Co., Inc.

P.O. Box 112730, Carrollson, TX 75011
(214) 245-4763

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

announces the properties of a variable . . . a definition also causes storage to be allocated."

If someone were to ask me to recommend a book to LEARN the C language from, I would suggest this book.

12. *C Primer Plus* — Waite, Prata, and Martin — SAMS Publishers

To be honest, I had a slight prejudice against this book before I opened the cover. I am NOT a big fan of Mitchell Waite's books; most of them talk to the reader as if he or she were in grammar school. For example, the *UNIX Primer Plus* , by the same authors, looks and reads like a Sesame Street product (i.e., it condescends to the reader), complete with pointless cartoons.

This book is really not bad, though. Most of the information is correct; there are some errors, however:

- The authors say that one dimensional arrays and pointers are different entities. The truth is that one dimensional arrays and single indirection pointers are the same internally. See page 210 of the bible (K&R) for verification.

- Waite and company say that comments are ignored by the C preprocessor. Again, the truth is that the C preprocessor actually strips comments out of C source files.

- Under the topic of macros vs. functions, the authors fail to mention that one of the disadvantages of using macros is that you cannot take the address of a macro but you can take a function.

- When the authors talk about Pointers and Multidimensional Arrays (page 372), they say,

```
int zippo[4][2];     /* 4 row, 2 column array of int */
int *pri;            /* pointer to int */
```

At this point the authors ask,

"Where does

```
pri = zippo;
```

point? It points to the first column, first row; i.e.

```
zippo == &zippo[0][0]"
```

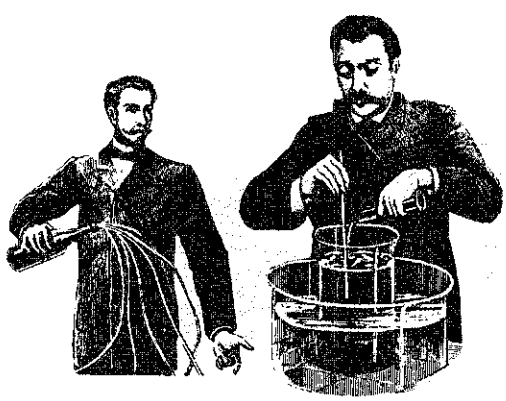
This is incorrect! The name of the 2-dimensional array, zippo, is NOT guaranteed to be a pointer to the zeroth row, zeroth column of the matrix. A rule in C states that the name of an array is a pointer to the zeroth element. Now, in this case, zippo is actually an array of arrays (remember, C is a lower level language) so that the zeroth element here is ANOTHER array. So, the name of the array (zippo) actually points to a WHOLE array. Obviously a pointer is ONE number, not a conglomerate; the actual address used as the pointer to the array is up to the compiler writer. The author of the compiler may elect to implement this pointer as the address of the last element of the zeroth array, instead of the address of the zeroth element of the zeroth array, as Waite, Prata, and Martin lead you to believe.

Some other sore spots include the treatment of the C preprocessor, which is too important to cover in just a few pages. Still another small point is that the function "malloc ()" (for memory allocation) is described so that we know how to dynamically allocate storage, but we don't find out how to deallocate memory. The answer is the function "free ()" (or "mfree()"), but the authors do not mention it.

The authors also confuse the topics of buffered input and unbuffered input.Their impression of buffered/unbuffered input may be correct for single user/tasking systems like PC-DOS on the IBM PC, but it is way off base for more complex operating systems like UNIX. A proper explanation of input buffers under UNIX would take an entire article alone. The authors should have at least mentioned this point.

Most other topics are explained in a fairly lucid manner. There is an excellent appendix that includes a program to generate sounds, written almost entirely in C for the IBM PC family. (Some students of mine have told me that programming in the C language itself causes programmers to generate sounds!)

The target reader for this book is the application programmer who already knows BASIC or FORTRAN or even Pascal. I would not recommend this book to seasoned assembly language programmers or experienced system programmers. All in all, I give this book a B+ grade. □



Famous Computer Personalities #12 and 13:
Marvin and Melton Woodcock performing early versions of data extract.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C SCREEN FUNCTIONS — PART ONE

by Ron Moak

From Ron Moak comes a listing of C screen functions. They contain assembly language functions that manipulate the screen and can be linked into C programs. Part One provides some basic functions to manipulate the cursor, initialize the screen, change mode, and read characters from the screen buffer.

Part Two, to be published in Volume 3, Number 5 (September/October 1985) of *Programmer's Journal*, contains more advanced functions for removing and restoring blocks of characters from and to the screen, printing lines with attributes, and scrolling selected portions of the screen.

These functions were written to be assembled with IBM's or Microsoft's Macro Assembler. The resulting object file may be linked directly into C code compiled with the Lattice compiler using the small memory model. Slight modifications to the source code will be required to support different compilers or memory models.

If you would like to make use of these functions, you may get the complete source, plus the object file, for $15.00. Please send a diskette with return postage and check to the following address:

Ronald Moak
Forest Computer Consultants
447 Brookside Dr.
Eugene, OR 97405

```
comment *------------------< HEADER >-------------------------------------

FILE:     SCRIO.ASM

TYPE:     ASM - Functions

PURPOSE:     Low level assembly language functions designed to interface
             with Lattice C.  These functions deal with screen input
             output.

COPYRIGHT:  Forest Computer Consultants (C) 1985

NOTES:     These functions may be linked into the Lattice C programs
           compiled to use the small memory module.

FUNCTIONS:  curget ( row,col,beg,end )
            curput ( row,col )
            curset ( type )
            scrcls ( )
            screol ( )
            scrinit( )
            scrmode( mode )
            scrread( )

---------------------------< EQUATES >--------------------------*

;  Video Functions

VIDEO       EQU  10H      ;VIDEO INTERUPT

SCR_MODE       EQU  0H      ;set screen mode
CUR_SET        EQU  1H      ;set cursor type
CUR_LOCATE     EQU  2H      ;set cursor position
CUR_GET        EQU  3H      ;get cursor type & location
READLP         EQU  4H      ;reading light pen position
SCR_PAGE       EQU  5H      ;select active page
SCROLL_UP      EQU  6H      ;Requests scroll up
SCROLL_DN      EQU  7H      ;scroll screen nown
CH_READ        EQU  8H      ;read a character from screen
CHA_WRITE      EQU  9H      ;write char and attributes
CH_WRITE       EQU  AH      ;write character only
SCR_STAT       EQU  15      ;find current screen status

;  Constants

ATTRIBUTE      EQU  7H      ;standard attribute character
BLANK          EQU  20H     ;blank character
SEGM_MONO      EQU  0B000H  ;Mono card segment
SEGM_COLOR     EQU  0B800H     ;Color card segment
ADDR_MONO      EQU  03BAH      ;Address of mono card
ADDR_COLOR     EQU  03DAH      ;Address of color card
SCR_CHAR       EQU  2000       ;Number of charcter stored on screen
CUR_LINE       EQU  1      ;two line cursor
CUR_HALF       EQU  2      ;half block cursor
CUR_BLOCK      EQU  3      ;full block cursor
CUR_COL_LIN    EQU  0607H      ;two line cursorcolor screen
CUR_COL_HLF    EQU  0407H      ;half block cursor color screen
CUR_COL_BLK    EQU  0007H      ;full block cursor color screen
CUR_MON_LIN    EQU  0A0DH      ;two line cursormono  screen
CUR_MON_HLF    EQU  070DH      ;half block cursor mono  screen
CUR_MON_BLK    EQU  000DH      ;full block cursor mono  screen
SCR_MONO       EQU  7          ;screen type 80 col mono
SCR_COLOR      EQU  3          ;screen type 80 col color
SCR_BW       EQU  2          ;screen type 80 col black & white
NUMCOL       EQU  50H      ;number of columns on the screen
NUMROW       EQU  19H      ;number of rows on the screen


;-------------------------< DATA SEGMENT >--------------------------

dgroup  group   data
data  segment para public 'data'
  public  SCR_TYPE
  assume  ds:dgroup

SCR_ROW        DB     ?     ;the row of the window
SCR_COL        DB     ?     ;the col of the window
SCR_TYPE       DB     ?     ;the type of screen
SCR_BUFF       DW     ?     ;the address of current screen buffer
ADD_6845       DW     ?     ;the address of the 6845 status port

data ends

;-------------------------< PROGRAM SEGMENT >-------------------------

pgroup group prog
prog segment byte public 'prog'
 public  curget,curput,curset,scratt,scrcls,screol,scrget,scrinit
 public  scrmode,scrprt,scrput,scrread,scrsrl
 assume  cs:pgroup

;-----------------------------------------------------------------
; name  curget -- get cursor position & type
;
; type  public
;
; synopsis  curget(&row,&col,&beg,&end);
;   short   *row;     row (0-24)
;   short   *col;     column (0-79)
;   short   *beg;     begin (0-13)
;   short   *end;     end (0-13)
;
; description  Returns the row, col and the beginning and ending scan
;line of the current cursor
;
;-----------------------------------------------------------------

curget proc   near
 push   bp
 mov    bp,sp
 xor    bx,bx
 mov    ah,CUR_GET  ;read cursor position & type
 int    VIDEO  ;BIOS video call
 xor    ax,ax
 mov    bx,[bp+4]  ;get &row
 mov    al,dh
 mov    [bx],ax  ;store it
 mov    bx,[bp+6]  ;get &col
 mov    al,dl
 mov    [bx],ax  ;store it
 mov    bx,[bp+8]  ;get &beg
 mov    al,ch
 mov    [bx],ax  ;store it
 mov    bx,[bp+10]  ;get &end
 mov    al,cl
 mov    [bx],ax  ;store it
 pop    bp
 ret
curget endp

;-----------------------------------------------------------------
; name  curput-- position cursor
```

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
 pop   bp
 ret
curget endp
;---------------------------------------------------------------
; name  curput-- position cursor
;
; type  public
;
; synopsis  curput(row,col);
;   short row;   0-24
;   short col;   0-79
;
; description  This function positions the cursor at the
;specified row and column.
;
;---------------------------------------------------------------
curput proc   near
 push   bp
 mov    bp,sp
 mov    ax,[bp]+4  ;get row
 mov    dx,[bp]+6  ;get col
 mov    dh,al
 xor    bx,bx  ;set page number = 0
 mov    ah,CUR_LOCATE  ;set cursor position function
 int    VIDEO  ;BIOS video call
 pop    bp
 ret
curput endp
;---------------------------------------------------------------
; name  curset -- set cursor type
;
; type  public
;
; synopsis  curset( type );
;   short type;  type = 1 - line cursor
; 2 - half block cursor
; 3 - full block cursor
;
; description  This function sets the cursor type.  Cursor types
;are automatically adjusted depending upon the type
;of screen installed.
;
;---------------------------------------------------------------
curset  proc   near
  push   bp
  mov    bp,sp
  mov    cx,[bp]+4  ;get beg
  mov    ch,cl  ;

  cmp    SCR_TYPE,7  ;test mono
  je    csmono  ;jump set mono cursor

  cmp    ch,CUR_LINE
  jne    csc2
  mov    cx,CUR_COL_LIN
  jmp    csinit
csc2:
  cmp    ch,CUR_HALF
  jne    csc3
  mov    cx,CUR_COL_HLF
  jmp    csinit
csc3:
  cmp    ch,CUR_BLOCK
  jne    cserr
  mov    cx,CUR_COL_BLK
  jmp    csinit
csmono:
  cmp    ch,CUR_LINE
  jne    csm2
  mov    cx,CUR_MON_LIN
  jmp    csinit
csm2:
  cmp    ch,CUR_HALF
  jne    csm3
  mov    cx,CUR_MON_HLF
  jmp    csinit
csm3:
  cmp    ch,CUR_BLOCK
  jne    cserr
  mov    cx,CUR_MON_BLK
csinit:
  mov    ah,CUR_SET  ;set cursor type function
  int    VIDEO  ;BIOS video call
cserr:
  pop  bp
  ret
curset  endp
;---------------------------------------------------------------
; name  scrcls
;
; type  public
;
; synopsis  scrcls( )
;
; description  Clear screen & position cursor to 0,0
;
;---------------------------------------------------------------
scrcls proc   near
 push   bp
 xor    cx,cx  ; upper left coord (0,0)
 mov    dx,2479h  ; lower right cord (24,79)
 mov    bh,ATTRIBUTE   ; clear attribute
 xor    ax,ax  ; zero al to scroll entire screen
 mov    ah,SCROLL_UP   ; scroll function
 int    VIDEO  ;
 xor    dx,dx  ; DH=0, DL=0 (DH=Row,DL=Column)
 xor    bh,bh  ; Screen # to use
 mov    ah,CUR_LOCATE  ; locate cursor function
 int    VIDEO  ;
 pop    bp
 ret
scrcls endp
;---------------------------------------------------------------
; name  screol -- clear to the end of the line
;
; type  public
;
; synopsis  screol
;
; description  This function will clear the line to the right of the cursor
;
;---------------------------------------------------------------
screol proc   near
 push   bp
 mov    ah,CUR_GET  ;see where we are
 int    VIDEO
 mov    cl,NUMCOL  ;calc how many chars left in line
 sub    cl,dl  ;number left
 xor    ch,ch  ;number of blanks needed
 mov    al,BLANK  ;write blanks
 mov    bl,ATTRIBUTE   ;normal attributes
 mov    ah,CHA_WRITE   ;write the blanks
 int    VIDEO
 pop    bp
 ret
screol endp
;---------------------------------------------------------------
; name  scrinit -- screen initialization routine
;
; type  public
;
; synopsis  scrinit(  );
;
; description  Initialization routine that is called at the beginning
;of the program to set up initial values
;
;---------------------------------------------------------------
scrinit  proc   near
 push   bx
 mov    ADD_6845,ADDR_MONO    ;load values
 mov    SCR_BUFF,SEGM_MONO    ;for mono
 mov    SCR_TYPE,SCR_MONO    ;all done
 mov    ah,SCR_STAT    ;get screen mode
 int    VIDEO    ;
 cmp    al,SCR_MONO    ;test mono
 je   siout
 mov    ADD_6845,ADDR_COLOR    ;load values
 mov    SCR_BUFF,SEGM_COLOR    ;for color
 mov    SCR_TYPE,SCR_COLOR    ;
 cmp    al,SCR_COLOR    ;if color screen then
 je   siout    ;all done
 mov    SCR_TYPE,SCR_BW    ;load for black & white
 cmp    al,SCR_BW    ;if b&w then all done
 je   siout
 mov    al,SCR_BW    ;force mode to 80 bw
 mov    ah,SCR_MODE    ;
 int    VIDEO    ;
siout:
 pop    bx
 ret
scrinit  endp
;---------------------------------------------------------------
; name  scrmode -- sets the screen mode for color graphics
;
; type  public
;
; synopsis  scrmode( mode );
;   short  mode;
;
; description  Sets the screen mode for color graphics monitors;
;this function has no effect on mono display boards
;
;   0 = 40 col black & white
;   1 = 40 col color
;   2 = 80 col black & white
;   3 = 80 col color
;   4 = 320 x 200 color graphics
;   5 = 320 x 200 black & white graphics
;   6 = 640 x 200 black & white graphics
;
;---------------------------------------------------------------
scrmode  proc   near
 push   bp
 mov    bp,sp
 mov    al,[bp+4]  ;new mode value
 cmp    al,SCR_MONO  ;test mono
 je   smout
 mov    SCR_TYPE,al  ;remember new values
 mov    ah,SCR_MODE
 int    VIDEO  ; set new mode
smout:
 pop    bp
 ret
scrmode  endp
;---------------------------------------------------------------
; name  scrread -- reads the character under the cursor
;
; type  public
;
; synopsis  ch = scrread( );
;   char  ch;
;
; description  Reads the character pointed to by the cursor and
;places the char into ch
;
;---------------------------------------------------------------
scrread  proc   near
 push   bp  ;save the registers
 mov    ah,CH_READ  ;code to read a character
 int    VIDEO  ;al is letter, ah=attributes
 or   al,al  ;blank returned instead of zero
 jnz    srl
 mov    al,BLANK
srl:
 mov    ah,0   ;kill the attributes
 pop    bp
 ret
scrread  endp
```

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

HOW TO CURE THE "DREADED LOCKED KEYBOARD" CONDITION

Or an Installation and User's Guide for the NMIEXIT Program.

by John W. Jackson

Ah yes, here we are, trying to debug that new program that is going to sell a million copies, and argh, the keyboard just locked up. Well, it's out there somewhere, looping out of control with all the interrupts turned off. If you could just look at that last file buffer, or examine a key location for clues. But alas no, it's power off, power on, and start single stepping. Perhaps you're not a programmer but have just purchased a wonderful, new, integrated spreadsheet and calorie-counting package. Each time you try to compute the bottom line on your dinner menu, it locks up. If only Alt-Ctrl-Del would work.

Well, have I got good news for you! For the price of a simple push-button switch, a very minor hardware modification to your IBM PC and the NMIEXIT program, your prayers will be answered.

HOW DOES IT WORK?

The NMIEXIT program takes advantage of a special line on the 8088, called the Non-Maskable Interrupt line. It is, as the name implies, non-maskable by normal software. This line appears on the IBM PC bus at pin A1 on every card in the IBM PC. This line is normally used by memory boards to signal another dreaded condition, the PARITY CHECK. In most other respects the NMI behaves much the same as any other 8088 interrupt. It has a vector at location 0000:0008, which normally points to a routine in BIOS that simply prints out the dreaded "PARITY CHECK" message and halts the machine.

A push-button switch may be wired from pin A1 of any PC board to ground. Pushing the switch will then simulate the parity check condition, and the PC will go running off to the BIOS NMI routine, print out the "PARITY CHECK" message, and halt. What good is that, you ask? Well, not much, but that is where the NMIEXIT program comes into play. Before you invoke the program that is giving you trouble, issue the NMIEXIT command. The command requires one operand, the name of any DOS com-type program. NMIEXIT makes itself resident along with the com program you supplied as an argument. The NMI vector is altered to point to the NMIEXIT program.

Now if you invoke your program and the keyboard locks up, simply push the button. The NMIEXIT program gets control. It first cleans house a little so that the display and keyboard are usable again. Next NMIEXIT displays all of the registers and flags as they were at the time the button was pushed. The com program that you provided as an operand to the command is now given control. If that program happens to be DEBUG.COM, you are in fat city. Display that buffer, dump that stack, look at those counters, and maybe you will find the clue.

The ability to specify any DOS com-type program to NMIEXIT is extremely flexible. For the nonprogrammer we talked about earlier who only wants to get his machine restarted, I have provided REBOOT.COM. This simple program jumps to the restart vector. You should never have to power off your machine again.

However, even this wonderful scheme has a weak spot. What if the rogue program wipes out NMIEXIT or the copy of DEBUG you made resident? There is a last resort, not pleasant though. For that last-resort case I have provided CASBAS.COM, a routine that gets you to cassette BASIC. Since cassette BASIC is in ROM, it cannot get wiped out. Now you can PEEK, MID$, and HEX$ yourself up a little dump routine to look at the remains of your problem program.

The DOS com program that you specify as an operand to the NMIEXIT command must contain a drive identifier and a path specification if they are other than the default. The extension COM must also be included. An example would be:

```
NMIEXIT C:\DEBUG.COM
```

Oh yes, it runs only under DOS 2.0 or later. DOS 1.1 users, upgrade. I do not know if NMIEXIT will work on any IBM compatible. I have tested it only on IBM PC and IBM PC-XT.

NOTES ON INSTALLING THE PUSH BUTTON

The installation of the push button is a simple task for anyone with even the most basic electrical and mechanical skills. However, if you have never had a soldering iron in your hands, enlist the aid of a friend who has.

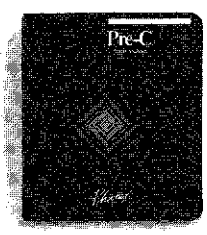
Here is a list of what you will need:

- a push-button switch, momentary contact, SPST, normally open
- a box to mount the button in
- a 2 ft. length of single conductor shielded cable
- a short length (several inches) of 30 or 40 GA wire
- an RCA phono jack and female connector that can be mounted on the faceplate of one of your PC boards

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Programmers' Pfantasies™ by Phoenix.

Phoenix makes programmers' dreams come true. With the best-engineered, highest performance programming tools you can find. A full line of MS-DOS® /PC DOS programs and utilities no other company offers. All designed to help you write, test and deliver the best programs possible. Top-of-the-line quality at a price you can afford.



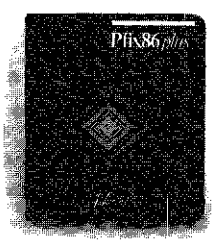
Finally, A Lint For MS-DOS.

Now you can get the full range of features C programmers working in UNIX™ have come to expect from their Lint program analyzer.

With Pre-C™ you can detect structural errors in C programs five times faster than you can with a debugger. Find usage errors almost impossible to detect with a compiler. Cross-check multiple source files and parameters passed to functions. Uncover interface bugs that are difficult to isolate. All in a single pass. Capabilities no C compiler, with or without program analyzing utilities, can offer. In fact, Pre-C outlints Lint, since you can handle analyses incrementally.

Pre-C's flexible library approach lets you maintain continuity across all the programs in your shop, whether you use Pre-C's pre-built libraries, pre-existing functions you already have, or some you might want to buy yourself.

Plus, you're not limited to one particular library, and Pre-C keeps track of all the libraries you're using to make sure that code calls them correctly. **$395.**

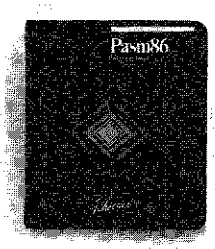


Still Fixing Bugs The Hard Way?

Pfix™86 Plus, the most advanced symbolic debugger on the market, eliminates the endless error searches through piles of listings. Locate instructions and data by symbolic name, using symbolic addresses. Handle larger, overlayed programs with ease.

An adjustable multiple-window display shows source, object code and data, breakpoint settings, current machine register and stack contents simultaneously. An in-line assembler allows program corrections directly in Assembly language. Powerful breakpoint features run a program full speed until a loop has been performed n times.

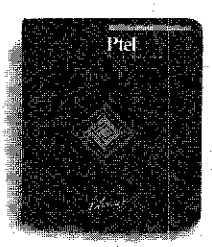
With a single keystroke you can trace an instruction and the action will be immediately reflected in source, object, data, stack, and register windows. Another key begins a special trace mode that executes call and loop instructions at full speed. Designed to work with both Plink™86 and MS® LINK linkage editors. **$395.**



Assemble Programs Twice As Fast.

Pasm™86 will assemble MASM files two to three times faster than MASM 3.0. Pasm86 supports 8086/88, 8087, 80186 and 80286 processors.

With Pasm86's built-in defaults, you can write code quickly since you won't spend hours learning all the control statements needed at the beginning of your program. You can define symbols on the command line. Decide whether you want error messages or not. And, put local symbols within procedures. **$295.**



Get The Lead Out Of Binary File Transfer.

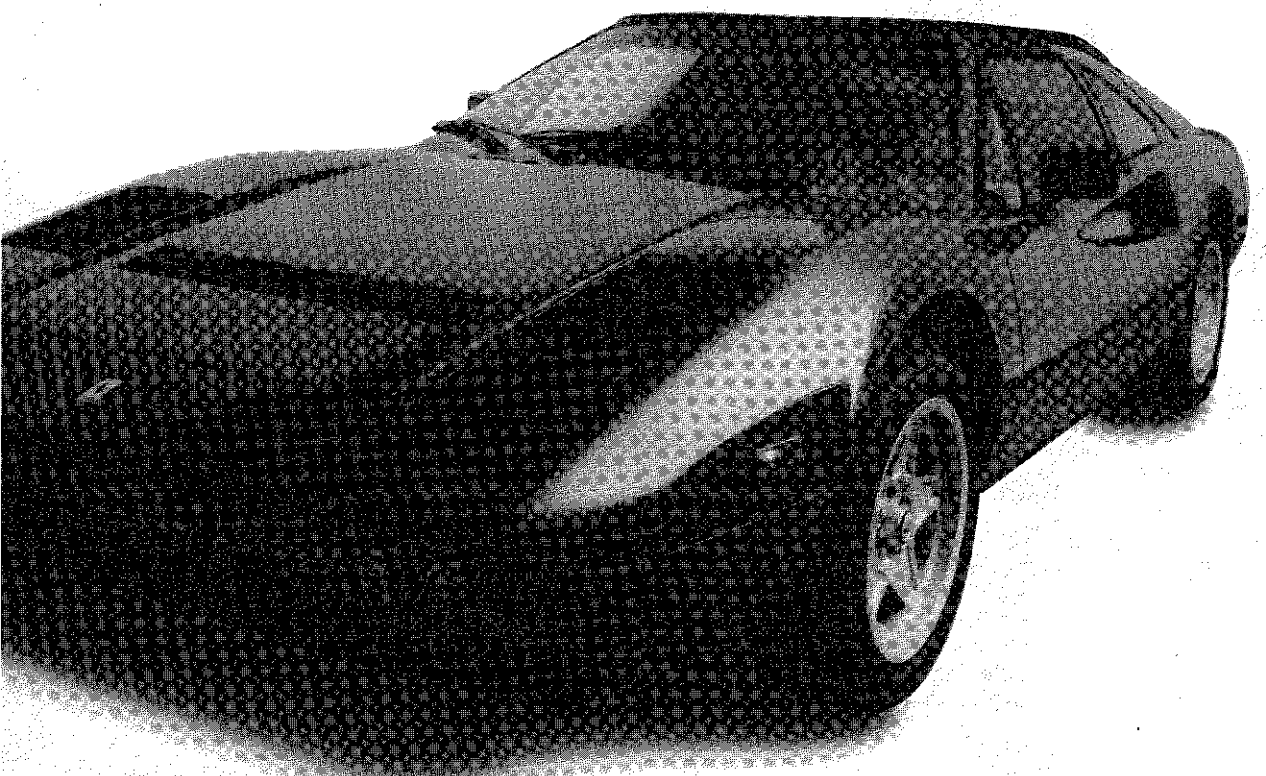
Ptel™ is the universal binary file transfer program for MS-DOS 2.0 or higher. You can move binary files fast and accurately. Upload or download groups of files from Bulletin Boards or remote computers. Move files between dissimilar machines and operating systems. Ptel's advanced binary protocol, Telink,™ offers better-than-Modem7 accuracy and performance. Faster transfer speeds. An on-screen update of error correction, blocks transferred, and transmission time.

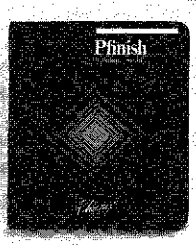
Includes popular Modem7 and XModem protocols. With checksum or CRC. Plus Kermit and ASCII. **$195.**

Special Thanks to Gasten Andrey of Framingham, MA.

Programmer's Pfantasies, Pre-C and Pfinish are trademarks of Phoenix Computer Products Corporation.
MS-DOS and MS are registered trademarks of Microsoft Corporation.
IBM is a registered trademark of International Business Machines Corporation.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

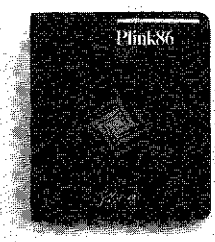




Maximize Your Program's Efficiency.

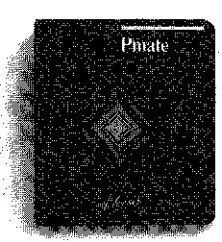
Pfinish™ delivers the fastest running programs possible. This performance analyzer lets you "zoom in" on the inefficient parts of your program. Whether written in Assembly language. C. PASCAL. FORTRAN. BASIC. Unlike profilers available today, Pfinish understands the structure of your program and reports the amount of activity and time spent in its subroutines or functional groups. Pfinish analyzes both overlaid and memory resident programs. Down to the instruction level. Reports are displayed. Stored on disk. Or printed out. In tabular form or histograms.

Do a dynamic program scan. Identify the most frequently executed subroutines. Find inefficient code that costs your program valuable time. Rank subroutines by execution frequency. **$395.**



Why Squeeze Your Program More Than You Have To?

The Plink86 overlay linkage editor brings modular programming to 8086/88-based micros. Write large and complex programs without worrying about memory constraints. Work on modules individually, link them into executable files. Use the same module in different programs. Change the overlay structure of an existing program without recompiling. Use one overlay to access code and data in other overlays. **$395.**



Why Work With A Primitive Editor?

More than a powerful editor, Pmate™ is a text processing language. An emulator of other editors. A language-specific editor for C, PASCAL, and FORTRAN. Pmate can even run in the background!

You get full-screen, single-key editing. Ten editing buffers. Horizontal and vertical scrolling. A "garbage stack" buffer. A built-in macro language with variables, control statements, radix conversion, tracing and 120 commands that you can group and execute with a single keystroke. **$225.**

Call (800) 344-7200. In Massachusetts (617) 762-5030.
Or, write.

Phoenix

Phoenix Computer Products Corp.
1420 Providence Highway Suite 115
Norwood, MA 02062

Outside the US, contact: Lifeboat Japan, Tokyo, Japan, Telex #2423296 LBJTYO, Telephone: 03-456-4101 • Repro Haganum (Stevis BV), Dinther, Netherlands, Telex #39581 REHAGNL, Telephone: 01-720-74543, 01-720-75543 • Memory Data, Sundbyberg, Sweden, Telephone: 46-8764-6700 • Roundhill Computer Systems, Marlborough, Wiltshire, UK, Telex #444453 AWARE G, Telephone: 44-467254675.

XT and AT are trademarks of International Business Machines Corporation.
UNIX is a trademark of AT&T Bell Laboratories.
Pasm86, Pfix86 Plus, Ptel, Telink, Pmate and Plink86 are trademarks of Phoenix Software Associates Ltd.
Intel is a registered trademark of Intel Corporation.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

All of these items are available at any Radio Shack store. Of course, if you are not into neatness, you can get by with only the button and wire.

First mount the button in the box, connect the shielded cable, and mount the male RCA phono jack to the end of the cable. If you have an ohm meter or continuity checker of some kind, it is good to check out the operation of the button at this time.

The next step is to select which board in your PC you are going to modify. Since the NMI signal appears on all boards, it does not really matter which one you choose. However, pick one that has room to mount the phono jack.

Remove the faceplate from the board, drill a hole, and mount the jack. Reinstall the faceplate.

Now let's find pin A1 and the ground pin on the board. Lay the board flat on a table with the component side up and the faceplate on your right. The A1 pin will be the first pin on the right end of the board fingers. The ground pin will be on the other side of the board, directly opposite the A1 pin. The plated paths will usually run from the connector fingers to a row of circular through-connects located just above the fingers. Some boards will have etched lables to help you locate the pins. The circular through-connects are the best place to solder and connect your wires. Use small-gauge wire (30 or 40) and run a connection from ground to the shell or outside of the phono jack. Run the other connection from pin A1 to the center terminal on the phono jack. Use solder sparingly on the board.

CHECKING IT OUT

Reinstall the board and put everything back together. Do not plug in your button yet. Power up the system and watch for smoke (just kidding). The machine should power up cleanly. If you get a PARITY CHECK error, you have shorted the NMI lead (pin A1) to ground somewhere. Now plug in your button. If you get a PARITY CHECK, there is a short somewhere in the cable or plug. Assuming all is still well and your system is up and running, now press the button. You should get a PARITY CHECK message and your system will lock up. This confirms that your button is doing what it is supposed to do. Now repower the system and invoke NMIEXIT, specifying DEBUG.COM as an operand. Remember to include any drive and path identifiers if required. NMIEXIT will give a confirmation message if all went well. Now hit the button. The screen will clear and NMIEXIT will display the registers and flags at the time the button was hit. At this time you should be in DEBUG. Try some of your favorite DEBUG commands. Entering the DEBUG "Q" command should return you to DOS.

HAPPY HUNTING.

For a copy, which includes a diskette, send $5.00 to:

NMIEXIT Program
Jackson Software
1729 Mayflower Dr.
Carrollton, TX 75007
(214) 492-1982

```
;** TODSTAMP 01-01-1980 23:56:59
;**************************************
;**************************************
;**
;** NMIEXIT
;** A PROGRAM TO ALLOW A RESIDENT COPY
;** OF COM FILES TO BE INVOKED
;** VIA THE NMI
;**
;**************************************
;**************************************
CSEG      SEGMENT
          ORG   100H
NMIEXIT   PROC  FAR
          ASSUME CS:CSEG,DS:CSEG,ES:CSEG
          ASSUME SS:CSEG
          JMP   A90
;**************************************
;**************************************
;**
;** DATA AREA
;**
;**************************************
;**************************************
DSN       DB    64 DUP(0)
HANDLE    DW    0
FSIZE     DW    0
OURSIZ    DW    0
APSP      DW    0

M1        DB    " NOT FOUND.",10,13,0
M2        DB    "** COM FILE INVALID.",10,13,0
M3        DB    "NMIEXIT VERSION 1.0",10,13,0
M4        DB    " NOW RESIDENT and ATTACHED TO NMI."
          DB    10,13,0
M5        DB    "** NO PARMS.",10,13,0

XTABLE    DB    "0123456789ABCDEF"

MYSS      DW    0
MYSP      DW    0
VDEBUG    DD    0

$AX       DW    0
$BX       DW    0
$CX       DW    0
$DX       DW    0
$SP       DW    0
$BP       DW    0
$SI       DW    0
$DI       DW    0
$DS       DW    0
$ES       DW    0
$SS       DW    0
$CS       DW    0
$IP       DW    0
$FLAGS    DW    0

OUTL1     EQU   THIS BYTE
          DB    "AX=0000",32,32
          DB    "BX=0000",32,32
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        DB      "CX=0000",32,32
        DB      "DX=0000",32,32
        DB      "SP=0000",32,32
        DB      "BP=0000",32,32
        DB      "SI=0000",32,32
        DB      "DI=0000",10,13,0

OUTL2   EQU     THIS BYTE
        DB      "DS=0000",32,32
        DB      "ES=0000",32,32
        DB      "SS=0000",32,32
        DB      "CS=0000",32,32
        DB      "IP=0000",32,32
ZFLG    DB      " XX XX XX XX XX XX XX XX",10,13,0

FLGTAB  DB      "OVNVDNUPEIDINGPLZRNZACNAPEPOCYNC"

;**************************************
;**************************************
A90:    CLD
        MOV     SI,OFFSET M3
        CALL    SUB2            ;PRINT GREETING
;**
;** GET THE UNFORMATTED PARM AREA AND MOVE IT TO DSN
;**
        SUB     CH,CH
        MOV     CL,DS:[80H]
        CMP     CL,0
        JE      A97

        MOV     SI,81H
        MOV     DI,OFFSET DSN

A95:    LODSB
        CMP     AL,32
        JNE     A96
        LOOP    A95

A97:    MOV     SI,OFFSET M5    ;NO PARMS
        CALL    SUB2
        JMP     RET

A96:    DEC     SI
        REP     MOVSB
        SUB     AL,AL
        STOSB

        MOV     DX,OFFSET DSN
        MOV     AL,0
        MOV     AH,3DH
        INT     21H             ;OPEN THE COM FILE
        JNC     A100

        MOV     SI,OFFSET DSN ;FILE NOT FOUND
        CALL    SUB2
        MOV     SI,OFFSET M1
        CALL    SUB2
        JMP     RET

A100:   MOV     HANDLE,AX
;**
;** SEE HOW BIG IT IS
;**
        MOV     BX,HANDLE
        SUB     CX,CX
        SUB     DX,DX
        MOV     AL,2
        MOV     AH,42H
        INT     21H
        CMP     DX,0
        JE      A110

        MOV     SI,OFFSET M2
        CALL    SUB2
        MOV     AH,9
        INT     21H
        JMP     RET

A110:   MOV     FSIZE,AX
        SUB     CX,CX
        SUB     DX,DX
        MOV     AL,0
        MOV     AH,42H
        INT     21H             ;RESET FILE POINTER
;**
;** COMPUTE THE SEG ADDRESS OF WHERE
;** TO CREATE DEBUG'S PSP
;**
        MOV     BX,OFFSET THEEND
        ADD     BX,15
        MOV     CL,4
        SHR     BX,CL
        MOV     AX,DS
        ADD     AX,BX
        MOV     APSP,AX
;**
;** CREATE A NEW PROGRAM SEGMENT PREFIX
;**
        MOV     DX,APSP
        MOV     AH,26H
        INT     21H
;**
;** NOW LOAD THE CODE
;**
        MOV     BX,HANDLE
        MOV     DX,APSP
        MOV     AX,DS
        SUB     DX,AX

        MOV     CL,4
        SHL     DX,CL
        ADD     DX,100H
        MOV     CX,FSIZE
        MOV     AH,3FH
        INT     21H
        MOV     AH,3EH
        INT     21H             ;CLOSE IT
;**
;** CHANGE THE NMI VECTOR TO POINT TO
;** THE NEWLY LOADED COM PROGRAM
;**
        MOV     DX,OFFSET NMI
        MOV     AL,2
        MOV     AH,25H
        INT     21H
;**
;** USE SETBLOCK TO SHRINK US DOWN
;**
        MOV     BX,APSP
        MOV     AX,DS
        SUB     BX,AX
        ADD     BX,128          ;+2K
        MOV     AX,FSIZE
        ADD     AX,15
        MOV     CL,4
        SHR     AX,CL
        ADD     BX,AX
        MOV     OURSIZ,BX
        MOV     AH,4AH
        INT     21H
;**
;** PROCLAIM OUR SUCCESS
;**
        MOV     SI,OFFSET DSN
        CALL    SUB2
        MOV     SI,OFFSET M4
        CALL    SUB2
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
;**
;** NOW MAKE US RESIDENT ...
;**
        SUB     AL,AL
        MOV     DX,OURSIZ
        MOV     AH,31H
        INT     21H             ;RESIDENT RC=0
;************************************
;************************************
RET:    MOV     AL,4
        MOV     AH,4CH
        INT     21H             ;BAD NEWS RC=4
;************************************
;************************************
;**
;** NMI ROUTINE
;** THIS IS WHERE CONTROL IS PASSED
;** WHEN THE NMI INTERRUPT OCCURES
;**
;************************************
;************************************
NMI     EQU     THIS BYTE
        ASSUME CS:CSEG,DS:NOTHING
        ASSUME ES:NOTHING,SS:NOTHING
        MOV     $AX,AX          ;SAVE AX FIRST

        IN      AL,62H
        TEST    AL,11000000B
        JNZ     B100
        MOV     AX,$AX
        IRET                    ;ITS NOT A PARITY CHECK

B100:   SUB     AL,AL
        OUT     0A0H,AL         ;DISABLE NMI

B110:   IN      AL,61H
        OR      AL,00100000B
        OUT     61H,AL          ;DISABLE I/O CHANNEL CHECK
        AND     AL,11011111B
        OUT     61H,AL          ;TURN EM ON AGAIN
        IN      AL,62H
        TEST    AL,01000000B    ;MAKE SURE FINGER
                                 IS OFF THE BUTTON
        JNZ     B110
;**
;** SAVE THE STATUS OF THE MACHINE ON ENTRY
;**
        MOV     $DS,DS
        MOV     $ES,ES

        MOV     AX,CS
        MOV     DS,AX
        MOV     ES,AX

        ASSUME CS:CSEG,DS:CSEG

        MOV     $BX,BX
        MOV     $CX,CX
        MOV     $DX,DX
        MOV     $SP,SP
        MOV     $BP,BP
        MOV     $SI,SI
        MOV     $DI,DI
        MOV     $SS,SS

        MOV     BP,SP
        MOV     AX,SS:[BP]
        MOV     $IP,AX
        MOV     AX,SS:[BP+2]
        MOV     $CS,AX
        MOV     AX,SS:[BP+4]
        MOV     $FLAGS,AX

        CLD

        PUSH    ES
        MOV     AX,40H
        MOV     ES,AX           ; ES -> BIOS DATA SEGMENT
        MOV     AH,0
        MOV     AL,ES:[49H]     ;CRT_MODE
        POP     ES
        INT     10H             ;RESET THE CRT CONTROLLER
;**
;** RESET THE KEYBOARD
;**
        IN      AL,61H
        OR      AL,11001000B
        OUT     61H,AL
        AND     AL,01111111B
        OUT     61H,AL
;************************************
;************************************
;**
;** DISPLAY THE REGS ON ENTRY
;**
;************************************
;************************************
        MOV     SI,OFFSET $AX
        MOV     DI,OFFSET OUTL1+3
        MOV     CX,8

D100:   CALL    SUB1
        ADD     SI,2
        ADD     DI,9
        LOOP    D100

        MOV     SI,OFFSET $DS
        MOV     DI,OFFSET OUTL2+3
        MOV     CX,5

D110:   CALL    SUB1
        ADD     SI,2
        ADD     DI,9
        LOOP    D110
;**
;** DISPLAY THE FLAGS
;**
        SUB     AL,AL
        MOV     AH,BYTE PTR $FLAGS ;LOW BITS OF FLAGS
        SHR     AX,1            ;CARRY
        SHR     AH,1
        SHR     AX,1            ;PARITY
        SHR     AH,1
        SHR     AX,1            ;AUX CARRY
        SHR     AH,1
        SHR     AX,1            ;ZERO
        SHR     AX,1            ;SIGN

        MOV     AH,BYTE PTR $FLAGS+1 ;HIGH BITS OF FLAG
        SHR     AH,1            ;THROW AWAY TRAP
        SHR     AX,1            ;INTERRUPT
        SHR     AX,1            ;DIRECTION
        SHR     AX,1            ;OVERFLOW

        MOV     CX,8
        MOV     SI,OFFSET FLGTAB
        MOV     DI,OFFSET ZFLG+1

F100:   SUB     AH,AH
        SHL     AX,1
        CMP     AH,0
        JE      F110
        MOV     BX,[SI]
        JMP     F120
F110:   MOV     BX,[SI+2]
F120:   MOV     WORD PTR [DI],BX
        ADD     SI,4
        ADD     DI,3
        LOOP    F100

        MOV     SI,OFFSET OUTL1
        CALL    SUB2
        MOV     SI,OFFSET OUTL2
        CALL    SUB2

;**
;** CLEAN THE PSP FOR DEBUG ...
;**
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        MOV   AX,DS
        MOV   BX,OFFSET THEEND
        ADD   BX,15
        MOV   CL,4
        SHR   BX,CL
        ADD   AX,BX
        MOV   APSP,AX
        MOV   ES,AX

        MOV   CX,164
        MOV   DI,5CH
        MOV   AL,32
        REP   STOSB

        MOV   BYTE PTR ES:[80H],0
;**
;** NOW INVOKE DEBUG
;**
        MOV   AX,100H
        MOV   WORD PTR VDEBUG,AX
        MOV   AX,APSP
        MOV   WORD PTR VDEBUG+2,AX

        MOV   MYSS,SS
        MOV   MYSP,SP

        MOV   AX,FSIZE
        ADD   AX,2048
        MOV   SP,AX
        MOV   AX,APSP
        MOV   ES,AX
        MOV   DS,AX
        MOV   SS,AX

        IN    AL,61H
        OR    AL,00110000B
        OUT   61H,AL
        AND   AL,11001111B
        OUT   61H,AL          ;ENABLE I/O CHAN & RAM
        MOV   AL,80H
        OUT   0A0H,AL         ;ENABLE NMI

        ASSUME DS:NOTHING,ES:NOTHING,SS:NOTHING
        STI
        JMP   DWORD PTR VDEBUG
;**
;** DEBUG WILL RETURN DIRECTLY TO DOS
;**
NMIEXIT ENDP

SUB1    PROC  NEAR
        ASSUME CS:CSEG,DS:CSEG

        PUSH  AX
        PUSH  BX
        PUSH  CX
        PUSH  DI

        SUB   AH,AH
        MOV   CL,4

        MOV   AL,[SI+1]
        SHR   AL,CL
        MOV   BX,AX
        MOV   AL,XTABLE[BX]
        STOSB

        MOV   AL,[SI+1]
        AND   AL,00001111B
        MOV   BX,AX
        MOV   AL,XTABLE[BX]
        STOSB

        MOV   AL,[SI]
        SHR   AL,CL
        MOV   BX,AX
        MOV   AL,XTABLE[BX]
        STOSB

        MOV   AL,[SI]
        AND   AL,00001111B
        MOV   BX,AX
        MOV   AL,XTABLE[BX]
        STOSB

        POP   DI
        POP   CX
        POP   BX
        POP   AX
        RET

SUB1    ENDP

SUB2    PROC  NEAR
        PUSH  AX
        PUSH  BX
        PUSH  SI

        SUB   BX,BX

SUB2A:  LODSB
        CMP   AL,0
        JE    SUB2B
        MOV   AH,14
        INT   10H
        JMP   SUB2A

SUB2B:  POP   SI
        POP   BX
        POP   AX
        RET
SUB2    ENDP

THEEND  EQU   THIS BYTE
CSEG    ENDS
        END   NMIEXIT
```

```
;** TODSTAMP 02-14-1985 19:55:42
;**
;** A COM PROGRAM TO JUMP TO
;** THE RESTART VECTOR
;**
CSEG    SEGMENT
        ORG   100H
REBOOT  PROC  FAR
        ASSUME CS:CSEG,DS:CSEG
        ASSUME ES:CSEG,SS:CSEG
        JMP   A100
VRESTART DW   0
        DW    0FFFFH
A100:   MOV   CX,20000
A200:   LOOP  A200
        JMP   DWORD PTR VRESTART
REBOOT  ENDP
CSEG    ENDS
        END   REBOOT
```

```
;** TODSTAMP 02-14-1985 19:57:04
;**
;** A COM PROGRAM TO INVOKE
;** CASSETTE BASIC
;**
CSEG    SEGMENT
        ORG   100H
CASBAS  PROC  FAR
        ASSUME CS:CSEG,DS:CSEG
        ASSUME ES:CSEG,SS:CSEG
        INT   18H
CASBAS  ENDP
CSEG    ENDS
        END   CASBAS
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

CALLING MS-DOS FROM TURBO PASCAL — IT'S A BREEZE

by Tom Swan

The built-in MsDos procedure in Turbo Pascal makes calling operating system routines a breeze. You can read the system date, check the DOS version number, change directory paths, list directories, or call any other DOS function — without once having to drop into assembly language.

The first step to calling an MS-DOS routine is to create a Pascal record containing the full set of 8088/86 registers. (To avoid any confusion here, from now on I'll use "MsDos" to refer to the Turbo procedure and "MS-DOS" to refer to the operating system.) Each 16-bit register is represented by an integer variable. The similar definition given in the Turbo manual, by the way, is missing the BP register and has the SI and DI registers printed in the wrong order. If you want to correct your manual, here is what the definition should be:

```
TYPE
   REGISTERS =
      RECORD
         AX, BX, CX, DX, BP, SI, DI, DS, ES, FLAGS : INTEGER
      END;
```

Experienced 8088/86 assembly language programmers know that the four 16-bit registers, AX, BX, CX, and DX, are each split into 8-bit halves. The 16-bit register AX, for example, contains the 8-bit pair, AH and AL. Unfortunately, if we use REGISTERS as defined above, there is no easy way to assign an 8-bit value to the AH half of AX, or to obtain the 8-bit DL part of DX. Rather than solve these problems by resorting to time-wasting divisions, we can instead create a new type that more closely resembles a real 8088/86 register:

```
TYPE
   A_REGISTER =
      RECORD CASE BOOLEAN OF
         TRUE : ( X : INTEGER );
        FALSE : ( L,H : BYTE  )
      END;
```

This is an example of the notorious, infamous use-at-your-own-risk, free union. The untagged CASE BOOLEAN declaration allows a variable of type A_REGISTER to be accessed as a 16-bit integer through the X field, or alternatively, as two 8-bit halves through the H and L fields. Because an integer is stored in byte-swapped order, the L field must be declared first. (The data type BYTE is predefined in Turbo Pascal as an 8-bit unsigned value.) Using this definition, we can now redefine the REGISTERS data type this way:

```
REGISTERS =
   RECORD
      A,B,C,D : A_REGISTER;
      BP, SI, DI, DS, ES, FLAGS : INTEGER
   END;
```

With this new definition, registers A, B, C, and D can be used as either 16-bit values, or as pairs of 8-bit values. For example, to assign the decimal value 1024 to register CX, you could use the following statement. Assume there is a variable R of type REGISTERS as defined above:

```
R.C.X := 1024;
```

Notice that CX (without the period) would not work. X is a field in register C, which is itself a field in the variable R. These are simulated machine registers, remember, not the real things. If you forget the period, though, don't worry, the compiler won't forget to complain. To assign 8-bit values to the high or low part of a register, use the H or L fields instead of X. For example, the following statements assign decimal 255 to register DH, and hexadecimal 1A to AL:

```
R.D.H := 255; R.A.L := $1A;
```

Turbo Pascal allows literal hexadecimal values to be declared as shown above. Any number preceded by a dollar sign ($) is interpreted by the compiler as a hexadecimal value. Because the DOS manual lists all function calls and many parameters in hex, this is an expecially handy feature when calling MS-DOS routines.

The same example can be shortened by using a WITH statement. To demonstate how, the statement below displays the values of the four 16-bit registers A, B, C, and D. Using WITH eliminates having to type R.A.X, R.B.X., etc. The :8 is a formatting command signifying an eight-character-wide column:

```
WITH R DO
     writeln( A.X:8, B.X:8, C.X:8, D.X:8 );
```

Be careful when using with in this way. In an earlier test program I had declared a global array variable named A. Because the REGISTERS variable R also has a field named A, the compiler became confused about which A I was referring

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

to inside the WITH statement. (I was using a function to find the address of A, and the compiler returned the address of the register field A instead of the array A, which is what I wanted.) In that test, not listed here, the program compiled correctly, but wouldn't run. I wasted an hour tracking that dumb bug down. This is always a problem when using WITH, but can be avoided by inventing field names different from any other identifiers. I mention all this as a caution about the REGISTERS data type. Don't use global variables named A, B, C, or D in your programs, or you might run into the same problem.

CALLING MS-DOS ROUTINES

I like to keep the definitions for both the A_REGISTER and REGISTERS data types in a separate "include" file named MSDOS.INC. The examples that follow expect to find this file on your compiler disk. To add the definitions to your own programs, put the following line anywhere in your global declaration area:

```
{$I MSDOS.INC }
```

After we include the A_REGISTER and REGISTERS type declarations, there are two steps to calling an MS-DOS routine. First, assign all required values, including the DOS function number as described in the DOS manual, to appropriate registers. Don't forget the function number! Some of the DOS routines write to the disk, and you wouldn't want to do that by accident. Next, execute the MsDos procedure, passing the REGISTERS variable as a parameter. As an example of how to do this, the following program prints the current MS-DOS version number:

```
PROGRAM version;

{$I MSDOS.INC }

VAR r : registers;

BEGIN
   r.a.h := $30;
   MsDos(r);
   writeln( 'MS-DOS Version = ', R.A.L, '.', R.A.H )
END.
```

The program works by assigning the DOS function number, hexadecimal 30, to the H field of register A. After that, DOS is called by passing register variable R to the MsDos procedure. By looking up function hex 30 in the DOS manual, we find that the major revision number is returned in register AL, and the minor number in AH. Following the call to MsDos, these numbers are displayed in a write1n statement. When executed on my XT, the program prints

```
MS-DOS Version = 2.10
```

Reading the system date is just as easy, as the next program demonstrates. Again, the DOS function number, this time hex 2A, is assigned to register AH before MsDos is called. Function 2A returns the month in DH, the day in DL, and the year in CX. If we use WITH to save typing, these registers are displayed in a single write1n statement. I used the MOD operator to display "85" instead of "1985," which works, assuming I've managed to get this piece published in the twentieth century.

```
PROGRAM getdate;

{$I MSDOS.INC }

VAR r : registers;
```


TallScreen™ TAKES THE DRUDGERY OUT OF DOS

- 1000 lines of screen output saved to scroll through for review and editing
- Commands and directory paths automatically saved for quick recall again and again
- Screen output and commands edited with standard, powerful, easy-to-learn and easy-to-use word processor techniques
- Screen output saved to file or printer, or frozen anywhere on the screen
- DOS commands help file included for quick reference
- Keyboard macros supported similar to the IBM Personal Editor.

For the IBM PC®, PCjr, XT, AT, or compatibles. Works with DOS 2.00 and above. Works with PROKEY™ and SIDEKICK™. Compatible with most mouse systems. Next day shipping on VISA/MC. MD & NM residents add sales tax.

$49.95
INTRODUCTORY PRICE

"TallScreen has completely changed the way I use DOS."
Bob Beilstein, PC user

"A godsend for power users."
Hanno Hinsch, Author of TELIOS

"This is the way DOS should have been designed in the first place!"
Rex Swain, Technical Director Parallax Systems

"Just recalling screen output and past commands alone is worth the price."
Edward T. Garner, VP Finance ARIES Group

For orders or information see your local dealer or call 800-522-0290

QUALITAS, INC., P.O. BOX 3AK/UPB
LAS CRUCES, NM 88003 505-522-0290

QUALITY PROGRAMMING TOOLS
QUALITAS™

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
BEGIN
   r.a.h := $2A;
   MsDos(r);
   WITH r DO
      writeln( 'Date is ', d.h, '/', d.l, '/', c.x mod 1900 )
END.
```

ACCESSING PASCAL VARIABLES

So far we have seen two DOS functions, get version and get date, that take and return their data in registers. There are other functions that access data in memory locations that are addressed by registers. To use these functions, you have to declare and then locate a Pascal variable in memory and pass its address to DOS. This is easy to do in Turbo Pascal with the two built-in functions, OFS (offset), and SEG (segment).

```
Program chdir;

{$I MSDOS.INC }

VAR r : registers; s : string[80];

BEGIN
   write( 'New Path? ' ); readln( s );
   s[ length(s)+1 ] := chr(0);
   WITH r DO
   BEGIN
      a.h := $3B;
      d.x := ofs( s ) + 1;
      ds := seg( s )
   END;
   MsDos(r);
   IF r.a.l <> 0
      THEN writeln( 'Path not found' )
END.
```

The listing immediately above demonstrates how to pass the address of a Pascal variable to DOS. The program prompts for a new directory path name, and if that path exists, sets the current directory to the sub directory at the end of the path. Although this action duplicates the CHDIR (or abbreviated CD) DOS command, you could use the same method inside a program to let users change directories without having to first exit to DOS. (My pet peeve with WordStar, to cite a notable example, is the lack of this ability.)

When you run this program, you'll be asked for a path name, which is stored in string variable S. Because the DOS function for changing directories requires an ASCIIZ string — a string ending with a null character — the Pascal string must be converted to the correct format. This is done in the second line following the BEGIN by adding a null character after the last character entered. I assume you will not enter 80 or more characters — although a better program would test for this condition

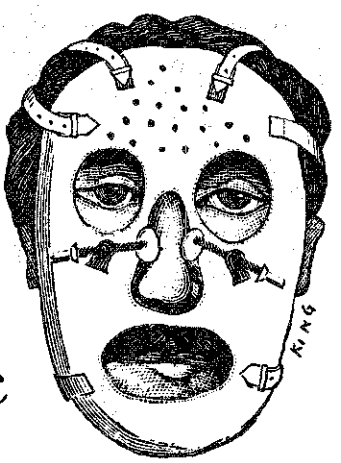
In the WITH statement, observe how functions OFS and SEG are used to find the address of the string variable. After we assign the DOS function number hexadecimal 3B to register AH, the offset and segment address of S is assigned to registers DX and DS. Because strings in Pascal begin with a length byte, the offset address is increased by one to locate the first character in the string. Finally, MsDos is called to change directories. If the path is not found, DOS returns an error code in register AL, tested here in an IF statement just before the program ends.

FLAG REGISTERS

Although there is supposed to be a provision for accessing single-bit flag registers after calling an MS-DOS routine, it's given me nothing but trouble. In the REGISTERS type definition, the FLAG field is supposed to contain the values of all flag registers upon return from DOS. Something is returned, but I don't think the flags are correct. However, because there is only one DOS function, number 6, direct console I/O, that passes a flag back as a parameter (the ZF flag in this case), this problem is not likely to cause any serious trouble — unless you want to write Pascal "filter" type programs that use the standard, and possibly redirected, input and output files. A helpful Borland technical representative wasn't aware of any bug reports, but also said he had no data at all on accessing the flags following a call to DOS. Still, I was unable to get DOS function 6 to work with the MsDos procedure. You should be able to use some combination of functions 1, 2, 7, and 8 to do the same thing, so this is no great loss.

Having a quick and easy access to MS-DOS is a welcome feature of Turbo Pascal. Other versions of the Borland compiler provide similar DOS procedure calls to CP/M-80 and CP/M-86 operating systems, a feature not found in many other high level languages. □

Tom Swan is the author of several computer books, including Mastering Turbo Pascal, *published with Turbo Pascal source code diskettes by Hayden Book Company. His newest book,* Pascal Programs for Data Base Management, *is due out in early 1986. Readers may write directly to the author at P.O. Box 206, Lititz, PA 17543.*



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

DOS 2.0 SORT AND FIND FILTERS

by Bernard H. Robinson, Jr.

This discussion of DOS 2.x filter commands includes much information that I have not seen in print elsewhere — information derived from experimentation with the commands. This all started when I was unable to find in the DOS 2.0 manual all of the information I needed for an application of the SORT filter to a certain job. This led to experimentation with the SORT filter and on to working with the other filter commands as well. This little article is the final result. All of the work was done on an IBM PC using DOS 2.0, but I believe that much of it is applicable to DOS 2.1 and possibly DOS 3.0.

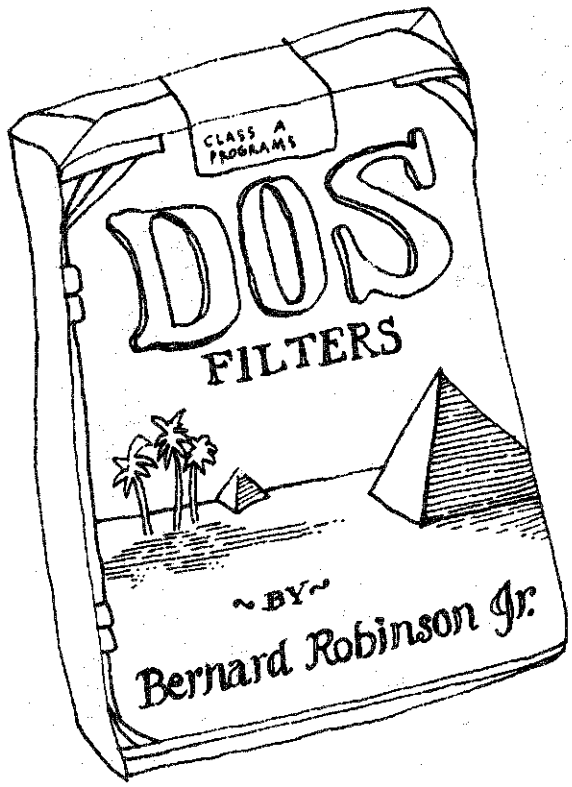
INTRODUCTION TO THE FILTERS

Chapter 10 of the IBM DOS 2.0 manual, "Advanced DOS Commands," covers redirection of standard input and output devices, piping of standard input and output, DOS filter commands, and some other topics. The term "filter" is applied to programs or commands that accept data from a device, process the data, and output the processed data to a device. The input comes from the standard input device (keyboard), and the output goes to the standard output device (CRT screen) — unless other sources and/or destinations are specified, usually by the redirection and piping capabilities of DOS 2.x. The three filter programs described in Chapter 10 are the FIND, MORE, and SORT commands.

All three of the filter commands may be called line-oriented commands in that they consider a complete record to be a stream of ASCII data that is terminated by a carriage return and a line feed (CR-LF). For example, when the FIND filter is asked to "FIND" a given ASCII string, it returns all lines in the file containing the given string anywhere in the line and any line returned is given in its entirety. The "/N" option of the FIND command causes the command to prefix each output line with its relative line number in the input file. These features of the FIND command make it suitable for searching a source program file in ASCII format for given search strings and reporting the location in the file of each occurrence of the search string.

In similar fashion, the SORT command regards a stream of data terminated by a CR-LF as one complete record. This line orientation of the filters is important in determining whether a given file can be filtered by them. The SORT filter will be discussed first. Much of what is said about the SORT filter also applies to FIND and SORT, and they will be discussed mainly in conjunction with their use with the SORT filter.

The SORT filter is a fast program, especially when used on a RAM disk. It is most suitable for sequential access files that are



to be sorted on only one field, which is the first field in each line of the file. If more than one data field is to be a sort field, the file must be formatted into neat columns, at least for all fields that are to be sort fields. See the discussion of the "/+nn" option of the SORT filter below. Since sequential access files allow data items to be of variable length, many sequential access files do not have the data items formatted into neat columns. This may be considered a limitation of the SORT filter.

The output produced by the DOS directory command is a good example of a file format for which the SORT command has much flexibility, so the sorting of directories will be used extensively for examples in the discussion that follows.

Perhaps I should state up front that if you are setting out to develop a flexible filing system, you will probably not want to depend upon the DOS SORT and FIND filters. Even a relatively inexpensive data base program, such as Jim Button's PC-File III, has much more convenient and flexible sorting and data finding capability for most applications. Nevertheless, I feel that it is worth the effort to become acquainted with the DOS filters. Most often they will be useful as spur-of-the-moment solutions to small problems, such as the following.

Recently I had a disk that served me as a backup disk for several programs, and I wanted to make a quick check to see whether the backup disk files all showed the same date and time as those on the primary disk. Each disk contained many

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

files that were not present on the other disk. The following sequence of commands made the job easier:

```
DIR A: >DIR.TXT
DIR B: >>DIR.TXT
SORT <DIR.TXT | MORE
```

The first command wrote the primary directory to a disk file; the second appended the backup directory to the same file (note the use of two " > " symbols, which requests APPEND; with only one " > ", just the second directory would be in the file). The third command sorts the combined directories by file name and outputs the sorted file to the screen through the MORE filter, one screen-full at a time. The sorting places files of the same name next to each other in the list, making it easy to compare the dates and times of the files.

Some of the filter commands use scratch files on the disk as they do their work. Generally, the filter programs will run faster if you copy FIND.EXE, SORT.EXE, and MORE.COM onto your RAM disk and make it the current default disk while you're using the filter commands. Sometimes while processing directories with filter commands, you may see one or more file names, such as %PIPE1.$$$, that you do not recognize. These are the scratch files being used by the filters, and they will be deleted automatically when the command is completed.

FILTER COMMAND SYNTAX DIFFERENCES

One of the petty annoyances of the DOS filter commands is the slight differences in syntax among the commands. The SORT and MORE filter commands require the "<" redirection symbol on the input file specification, but the FIND command does not. I have had the computer "die" when I forgot to put the "<" symbol on the input file name for the SORT command, but control-break got me back to DOS.

A filter command must have a space in front of the redirection symbol ("<" or ">"), but there is no need to have a space following it. When the output of a command is to be piped into a filter, there should be a space before and after the "|" pipe symbol, and the filter input file specification should be omitted. An example of this was shown above, and more examples are given below.

The FIND command has options not provided for the other commands. The "/V" option causes lines NOT containing the search string to be output. The "/C" option causes the output to be only the count of the number of lines containing the search string without the lines themselves. The "/N" option causes the line number of each output line to be added to the beginning of the output lines.

The "/R" option of the SORT command causes the sort to be reversed so that "Z" comes before "A". The "/+n" option on the SORT command is used to specify the column in the data for starting the sort. It would be handy to have the same option for the FIND filter, but none is provided. The "/+n" option of the SORT command cannot be used unless the data item field you want to sort on always starts in the same column in each record of the file, which is why I mentioned above that the SORT command is most flexible on files that have their data items formatted into neat columns. Ways to achieve this file format will be suggested later, but for now, let's continue to consider examples of the use of the filter commands on the neatly formatted disk directory command output.

SORTING AND FILTERING DIRECTORIES

The following merely demonstrates the use of filter commands and is not intended to be a replacement for the more flexible directory utility programs that are available. If we follow the example in the DOS manual and type DIR | SORT, the current default directory is sorted according to the beginning columns of each line; that is, by the program name, and the result is output to the console screen. But suppose we want to sort the directory by file extension. That is easy — we type DIR | SORT /+10 to start the sort on column 10, the start of the file extension.

Sorting by size of the files is also easy since the size of each file starts in column 14. Suppose we want to sort by the date the file was last updated. Although the date starts in a fixed column, we have a problem because the first part of the date is the month, not the year. Although it is a bit clumsy, we can do the date sort the following way. Say, for example, that all the files in the directory are dated 1983 through 1985. The following command sequence will do the trick:

```
DIR | FIND "-83 " | SORT /+24 >DIR.TXT
DIR | FIND "-84 " | SORT /+24 >>DIR.TXT
DIR | FIND "-85 " | SORT /+24 >>DIR.TXT
```

In the first line above, the FIND filter outputs only the directory entrys containing the 1983 date, the SORT portion of the command sorts these entries according to month and day, and finally, the sorted output is written in the file DIR.TXT. The next command does the same for 1984 entries and appends the sorted list to the sorted 1983 entries already in the DIR.TXT file. Finally, the third line completes the job by doing the same for 1985 entries.

If you make the assumption that all files in your directories are dated 1981 through 1985, you could write a straightforward batch file containing five lines to sort directories by date. The most important point here is the way the piping feature of DOS allows the various filters to be combined to get a desired result.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Periscope ... A Direct Hit!

"A great debugger ... If you've been frustrated by programs that don't behave the same for the debugger as when running alone, or that crash altogether, you should check out Periscope ..."

PC REPORT, Boston Computer Society

"It works, and works well!! In the first day of use I finished up two weeks of problems!! Really excellent!!!"

Peter Loats, Periscope User

Periscope's differences hit home! Its breakout switch gives you control, even when interrupts are disabled. Periscope's unique power helps you solve the difficult problems. Debug device-drivers, memory-resident, and non-DOS programs.

Great for straightening out confused C pointers! Using its breakpoint power, you can stop on reads and writes to ranges of memory. Stop on registers, words, and bytes, too.

It's tough. Periscope's 16K RAM board is write-protected. Runaway programs can't touch crucial debugger code! Examine the system, safely, at any time: Periscope saves the interrupt vectors and buffers it uses, then restores them when done. It uses ROM BIOS calls for functions except file access, so DOS can't clobber itself.

It gets you moving. It's commands are similar to Debug's, so you'll master it quickly. On-line help is there when you need it. You can define the windows you want; you can design easy-to-read memory displays. Periscope supports C, Assembler, BASIC, and Pascal. Comprehensive symbol support gives you a roadmap through memory tying back to your source.

It's risk-free. Periscope is backed up by a 30-day, money-back guarantee! The board is warranted for a year. Technical Support? You get the best there is ... direct from Brett Salter, Periscope's author!

It requires. An IBM PC, XT, AT, Compaq, or close compatible; PC-DOS, 64K RAM; a disk drive and an 80-column monitor. With two monitors, Periscope's screen is displayed on the monitor not in use. With one monitor, a keystroke switches screens.

It's power you can afford. At $295, you can afford Periscope's professional power. The system includes the memory board, breakout switch, debugger software, manual, and quick reference card.

Order now! Call (800) 421-5300, ext. R96, 24 hours a day to order with MasterCard/VISA. Demo disk is $5. Questions? Please call (404) 256-3860.

Get your programs up and running;

UP PERISCOPE!

Data Base Decisions/14 Bonnie Lane | Atlanta, GA 30328

MBASIC Programmers

Save **Time & Money** with

HELP MESSAGE SYSTEM™

Minimize documentation! Create and edit, context sensitive, interactive help and/or tutorial functions. Turns your text information into callable libraries. Overhead to your program is insignificant.

B-PLUS TREE SUB-SYSTEM™

Write your program without worrying about the handling of index files. Input/output is almost instantaneous, independent of file size. Direct and sequential accessing, variable length keys, multiple non unique keys and multiple keys per record.

SCREEN BUILDER SYSTEM™

Using either the built in editor or your word processor, create edit fields of any length anywhere on the screen with full screen editing within each field and full screen movement from field to field. The generated code is ready to insert in your program.

EZ SCREEN™

An Assembler language routine embedded in a basic subroutine to perform screen handling and windowing functions when called from a Basic program.

Priced As Low As **$59** Each

30 DAY $ BACK GUARANTEE
NOT COPY-PROTECTED
NO ROYALTY

Any 1 Program $79
Any 2 Programs $138 ($69 each)
Any 3 Programs $198 ($66 each)
All 4 Programs $236 ($59 each)

Visa, Mastercard, Checks or C.O.D.
CA residents add 6.5% sales tax
No shipping charges

STERLING CASTLE

702 Washington Street Suite 174
Marina Del Rey, California 90291
(213)306-3020

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Sometimes two or more FIND commands in tandem are required to get a filtering job done. For example, suppose we want to find only ".COM" files dated in the month of April 1985. The command could be

```
DIR *.COM | FIND "-85 " | FIND " 4-" >DIR.TXT
```

The following command:

```
DIR | FIND /V " BAS " | FIND " 4-03-85" >DIR.TXT
```

would output all files which are dated 4-03-85 that are NOT BASIC files (/V option on the first FIND).

The following command:

```
DIR \*. | FIND "<DIR>" >DIR.DAT
```

will send a directory listing of all subdirectories in the root directory to the file DIR.DAT. This file could then be opened by a BASIC program, and the first 8 characters of each line of the file could be isolated. This would give the BASIC program the name of each subdirectory of the root directory. From this starting point, a BASIC program could be written to perform some useful function such as searching subdirectories for a file or generating a directory listing of all files on the disk together with the name of the subdirectory they are in.

It is true that just the "DIR *." part of the above command, without the 'FIND "⟨DIR⟩" ' part, would have given a directory listing of all subdirectories in the root directory, but the listing would also have included the names of any ordinary files that have no extensions, which we do not want in this case.

HELPFUL TRICKERY

The examples above illustrate that it may be necessary to study the file format and take advantage of subtle features of it in using the filter commands. For instance, even though the FIND command does not have a "/+n" option to allow us to select the starting column, we could isolate the year part of the date by specifying the leading "-" character and the trailing space in the FIND search string. If we were designing the format of a file to be handled with the filter commands, we could have an easier time of it. We could deliberately insert odd characters such as @, #, and &, or even some of the upper 128 extended ASCII characters as part of the field we want to FIND on. As an example, suppose we want to designate each record as being in one of 50 different categories and want to use the FIND command to pull out of the file all records belonging to one or more of the categories. We could put a data item in the file in the form of "&nn" where nn is the numerical classification of the record and "&" is just a symbol included to make it possible for the FIND command to distinguish this number from any other in each record. We could then use "&35" as the search string in the FIND command to output all records classified as type 35 records, for example.

RESTRICTIONS ON TYPES OF FILES

So far we have barely mentioned an important aspect of the filters, the types of files they may be used on. Files must meet the following requirements if they are to be handled correctly by the filters:

1. Each complete record in the file must be terminated by a carriage return followed by a line feed (CR-LF). Each record can be thought of as a line of data. There must not be a carriage return in the record itself. Even though it is permissible in sequential files, under some circumstances, to include a carriage return in a string that is enclosed in quotation marks, a carriage return within a record will prevent proper sorting — even if it is within a string enclosed in quotation marks.

2. The records must not contain any embedded extraneous record terminators or (especially) sequential file end-of-file characters, chr$(26). This requirement rules out many random access files since one of the reasons for using random access files is to take advantage of their ability to store numerical data in binary format as opposed to ASCII character format. Numbers in binary format are very likely to have embedded in them bit patterns which the filters will interpret as end-of-file markers or record terminators. If random access files do not contain numerical data in binary format and meet the other requirements discussed below, they can be handled by the filters. See "Formatting Files for Use with the Filters," below.

3. The file must be terminated with a standard sequential end-of-file character, chr$(26). This requires a special operation for random access files. See the formatting discussion below.

4. The maximum length of a record that can be sorted is not given in the DOS manual. It appears that records at least as long as 255 characters can be sorted. I have not tried any longer records. The manual states that the maximum length of a file that may be sorted is 63K.

To clarify the requirements a bit, let's consider a file of addresses. Each record might consist of the following data items: a name, a street address, a city, a state, a zip code, and a telephone number. If we want to sort the file with the SORT command, the file must be written to the disk with a carriage return followed by a line feed after the last data item in each record (the telephone number) — and with no CR-LFs within

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

the record. This can be achieved easily in a sequential access file by using only one WRITE # statement to write each full record to the file. Although it is more complicated to do, PRINT # and PRINT # USING statements could be used instead of one WRITE # statement (see the discussion of these statements in the BASIC manual).

In the example above of a sequential access address file, it would be easiest to sort on the first data item, the name. The names would be entered with the surname first so that the sort could be made by surname. You could sort on any other single data item by making it the first data item in the record. If you wanted to be able to sort on two or more data items of the record, say on either the name or the state, for example, it would be necessary to make both the name and the state start in predetermined columns of the file in each record so that the "/+n" option could be used on the SORT command to select which field to sort on.

FORMATTING FILES FOR USE WITH THE FILTERS

To avoid turning this article on filters into a tutorial on BASIC data files, I've deliberately kept the following discussion brief.

There are a number of ways to control the column position of a data item in the file. The data items for which position is critical are all of those from the beginning data item of each record to the last data item that will be used as a sort field. For all critical data items, a suitable maximum length must be determined, and each entry must be held within its field length. One could then use the TAB() function to position each critical data item within the "line" of each record. In setting up the file format, you must count as characters all of the data item separators (mostly commas and quotation marks) that must be written on the disk for sequential access files. See the discussion in the BASIC manual for the WRITE #, the PRINT #, and the PRINT # USING statements. The PRINT # USING statement can also be used to help control the starting column for critical data items, but it is a little tricky to use. To verify that the format design is correct, it is often helpful to write a few records on a file and then print the file to the screen using the DOS TYPE command since this will display all the commas and quotation marks inserted by the WRITE # statement.

RANDOM ACCESS FILES FOR USE WITH FILTERS

Random access files afford relatively tight control over the position of each field (data item) in the record through the FIELD statements. It has been suggested that if sorting is to be done on more than a couple of data items, random access files are probably the way to go. To meet the requirement of the filter commands that the end of each record be identified by a carriage return followed by a line feed, two character spaces at the end of the FIELD statement must be allotted to these two characters, and they must be properly filled — chr$(13) for a carriage return and CHR$(10) for a line feed. There must be no numerical data in binary format, so the following BASIC functions must not be used: MKI$(), MKS$(), and MKD$(). All numbers must be in ASCII format obtained by conversion from numerical data format with the STR$() function in the case of random access files or as converted by the STR$(), WRITE #, PRINT #, or PRINT # USING statements in the case of sequential access files.

The filters want all files to be terminated as sequential access files. After much experimentation, I have found that the easiest way to do this on a random access file is as follows: After the file has been closed as a conventional random access file, reopen it for APPEND as a sequential file and then close the file without writing anything to it. For example:

```
500 OPEN "XYZ.DAT" FOR APPEND AS #1
510 CLOSE #1
```

This will cause the correct file termination to be appended to file XYZ.DAT.

Random access files are often initialized with a file size (number of records) to accommodate many more records than are initially in the file. This could cause problems with the SORT command because all of the blank records would be sorted to the beginning of the file. Do not set up a file this way if you know you will be using the filter commands on it. If you already have a file like this, see the discussion below for a possible solution.

Another procedure often used with random access files is to write a specified character as the first character in each record to mark the record as available for use — both to initialize the records and to delete a record by just writing the specified character in the first character position in the file, actually leaving the rest of the "deleted" record in the file. A file like this can be handled by the SORT and FIND filters if the following steps are taken. Select as the character to be used to mark the record as deleted some character or character combination not allowed in the file elsewhere. You will then be able to include a FIND filter that will throw out all records that include this marker. For instance, if "@" identifies the record as deleted and is not allowed elsewhere in the file, you could write

```
FIND /V "@" XYZ.DAT >XYZS.DAT
```

which would provide an output file with all marked records really purged. Also a command like

```
FIND /V "@" XYZ.DAT | SORT /+10 >XYZS.DAT
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

would sort the file on column 10 while purging the file of all marked (deleted or blank) records. The same FIND /V "@" filter command could also be used to skip deleted records when searching with a FIND command.

SORTING ON NUMERICAL DATA FIELDS

All numbers in the sort field must be ASCII characters. The SORT filter treats numbers as merely ASCII characters and sorts by the ASCII value of each digit. The sort will be numerically correct only if the following conditions are met:

1. If the numbers were obtained from BASIC variables, they must have been converted with the STR$() function or by using the WRITE #, the PRINT #, or the PRINT # USING statement. The MKI$(), MKS$(), and MKD$() functions must not have been used in the conversion.

2. All numbers in the sort field must be positive numbers (no negative numbers).

3. The decimal point, whether expressed or just understood, must be in the same column for each number in the sort field. If decimal fractions are present, all numbers must have the same number of digits to the right of the decimal point. This means that when decimal fractions are present in the sort field, final zeros in the selected number of places to the right of the decimal point must be zero characters and not just omitted as BASIC does unless special steps are taken. The special steps could consist of using string commands to test and possibly correct each numerical string in the sort field or else resorting to the PRINT # USING statement (best, but a bit tricky). Common fractions (1/2, 1/8, etc.) are not allowed, and neither are numbers in exponential format (12.123e-8, etc.).

4. Any non-numeric characters in the sort field, such as the "-" symbols in the date field of directories, ":" symbols in time fields, decimal points, or "$" signs, must be in the same column in each record, and if they are in one of the records, they must be in all records.

If the above requirements are met, the sort should be numerically correct.

```
09/14/1752  2361222  THU  09/14/1752
12/31/1752  2361330  SUN  12/31/1752
01/01/1753  2361331  MON  01/01/1753
12/31/1799  2378496  TUE  12/31/1799
01/01/1800  2378497  WED  01/01/1800
02/28/1800  2378555  FRI  02/28/1800
03/01/1800  2378556  SAT  03/01/1800
12/31/1800  2378861  WED  12/31/1800
01/01/1801  2378862  THU  01/01/1801
02/28/1801  2378920  SAT  02/28/1801
03/01/1801  2378921  SUN  03/01/1801
12/31/1899  2415020  SUN  12/31/1899
01/01/1900  2415021  MON  01/01/1900
02/28/1900  2415079  WED  02/28/1900
03/01/1900  2415080  THU  03/01/1900
12/31/1900  2415385  MON  12/31/1900
01/01/1901  2415386  TUE  01/01/1901
12/31/1999  2451544  FRI  12/31/1999
01/01/2000  2451545  SAT  01/01/2000
02/28/2000  2451603  MON  02/28/2000
02/29/2000  2451604  TUE  02/29/2000
03/01/2000  2451605  WED  03/01/2000
12/31/2000  2451910  SUN  12/31/2000
01/01/2001  2451911  MON  01/01/2001
12/31/2099  2488069  THU  12/31/2099
01/01/2100  2488070  FRI  01/01/2100
12/31/2199  2524593  TUE  12/31/2199
01/01/2200  2524594  WED  01/01/2200
02/28/2200  2524652  FRI  02/28/2200
03/01/2200  2524653  SAT  03/01/2200
12/31/2300  2561482  MON  12/31/2300
01/01/2400  2597642  SAT  01/01/2400
02/28/2400  2597700  MON  02/28/2400
02/29/2400  2597701  TUE  02/29/2400
03/01/2400  2597702  WED  03/01/2400
01/01/2600  2670691  WED  01/01/2600
02/28/2600  2670749  FRI  02/28/2600
03/01/2600  2670750  SAT  03/01/2600
12/31/2799  2743738  FRI  12/31/2799
01/01/2800  2743739  SAT  01/01/2800
02/28/2800  2743797  MON  02/28/2800
02/29/2800  2743798  TUE  02/29/2800
03/01/2800  2743799  WED  03/01/2800
12/31/3199  2889835  FRI  12/31/3199
01/01/3200  2889836  SAT  01/01/3200
12/31/3999  3182029  FRI  12/31/3999
01/01/4000  3182030  SAT  01/01/4000
02/28/4000  3182088  MON  02/28/4000
```

□

WRITE IT DOWN

Nobody knows better than the readers of *Programmer's Journal* exactly the kinds of information, tips, and tricks that programmers can use. We want to hear what you have to say about code you've developed, products you've tried, the language(s) you favor, books you've read. We're interested in your views on general computing topics, too — artificial intelligence, commercial development, legal questions, the latest computer conference you attended — the way things are shaping up for programmers. Write it down and send it to

Programmer's Journal
P.O. Box 30160
Eugene, OR 97403

Send us your article or review on disk — standard ASCII text file, please — accompanied by a printout. Attach some information about yourself if you'd like that to appear, too. Expect to hear from us about your article within the month. Send us a stamped self-addressed envelope if you'd like us to return your disk and manuscript.

In addition to the satisfaction and prestige to be gotten from publishing in *Programmer's Journal*, you can make money, too — we won't make you rich, but we can help make you famous. Think about it . . . Write about it!

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

the SET Design Tools

THE SET TOOLS

review by Bill Salkin

System Engineering Tools, Inc.
645 Arroyo Drive
San Diego, CA 92103
(619) 692-9464

The SET tools, a group of utility programs, aid programmers with the software development and maintenance process. PATCH, the first tool reviewed, is a disk or file-patcher. The second tool, GXREF, generates a cross-reference of any source-code file. The third tool, DIFS, finds differences between two ASCII files. (With an add-on product "change cards" can be applied to the updated file to recreate the original file.) SCIL, the last tool reviewed, is a source-code librarian. It brings mainframe library-management capabilities to your PC.

Each tool performs as claimed in the user's manual. Error-handling is very good.

The SET tools run on a wide variety of computers and operating systems. I tested them on an IBM-PC computer under PC-DOS.

PATCH (price $79.00)

PATCH is a disk or file patcher. In file-mode, PATCH restricts its action to a single file. In disk-mode, any byte on the disk is fair game.

PATCH displays one 128-byte record at a time. For example, the first 128 bytes of the file "READ.ME" is shown below.

```
 Old File READ.ME        3 Filesize    30 3 Old Record     0 3 Byte    1

I f   y o u   c a n   r e a d     t h i s ,   t h e n   y o u   a
496620796F752063 616E207265616420 746869732C207468 656E20796F752061

r e   d r i v i n g   t o o   d a m n   c l o s e !   O l d
7265206472697669 6E6720746F6F2064 616D6D20636C6F73 65210D0A4F6C6420

c o l l e g e   p r o f e s s o r s   n e v e r   d i e .   T h
636F6C6C65676520 70726F6665737373F 7273206E65766572 206469652E205468

e y   j u s t   l o s e   t h e i r   f a c u l t i e s .   F
6579206A75737420 6C6F736520746865 697220666163756C 746965732E0D0A46

   If you can read this, then you are driving too damn close!  Old
   college professors never die. They just lose their faculties.  F

Esc - Return to record number      3F1 - Help (for next key pressed)
F2  - ASCII conversion table       3F3 - Create .BAK of current file
Arrows - position cursor           3<CR> - writes record
Home   - positions to first record 3PgUp - positions to record -1
End    - positions to last record  3PgDn - positions to record +1
```

The file name and filesize are shown on the top line, as is the current record-number (0) and the byte offset (1) from the start of that record. The filesize value (30) is in units of 128-bytes.

Typing a character in the ASCII area updates the Hex area. Typing two characters in the Hex area updates the ASCII display. The file itself is not modified by your changes until the Enter key is pressed.

PATCH can browse through files in a sequential or random manner (by typing a record-number value). Additionally, PATCH can search for a user-entered ASCII or Hex pattern.

The PATCH display used for the disk-mode is similar to the file-mode display except that the filesize field is replaced by a sector-number field.

GXREF (price $79.00)

GXREF is a cross-reference tool. It uses filter-files to include or exclude words from the cross-reference listing. Using GXREF without a filter-file causes every space-delimited alphanumeric string in the source code to appear in the GXREF listing.

Three filter-files are supplied with GXREF: one file for COBOL, one for FORTRAN, and one for C. The C filter is shown below.

```
-----------------------------C Filter---------------

-w0123456789
-x for while do BEGIN END if char struct int long float extern printf fprintf
-xgets fgets puts fputs getc fgetc putc fputc open close fopen fclose
-s/* */ -s' ' -s" "
-xFILE
-g_
-xbreak default return define case else endif exit ifdef sprintf read write
-cfinclude
-xscanf strcpy strncpy strcmp strncmp switch toupper isdigit typedef ungetc
-xungetch unlink unsigned

-----------------------------C Filter---------------
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

In the first line the "-w" option means that GXREF should include words containing numeric values, except if the numeric value occurs in the starting position of the word. So "x1" is included in the GXREF listing, but "1x" is not.

The "-x for" option indicates that the word "for" is excluded from the GXREF listing. The words "while," "do," "if," and "char" — the standard C keywords — are also excluded from the listing. Many standard C functions such as strncpy are also excluded. (I don't know why BEGIN and END appear in this C filter.)

The "-s" or skip option means that any words found between the following start and stop characters are skipped or excluded from the listing. The expression "-s/* */" means to skip any characters found between "/*" and "*/". In other words, the contents of C comment statements are ignored.

The "-g_" option includes the underscore character as a Good character. Words containing underscores are treated as being distinct, so "_goes" and "goes_" are different. Without the "-g_" option, GXREF ignores the underscore, and "_goes," "goes_," and "goes" are regarded as the same word.

When you use other directives (not shown above), words appearing inside parentheses after a FORMAT or "printf" statement can be excluded from the listing.

It must be emphasized that the C filter-file (or any other filter) can be modified using a text editor. If you don't like it, change it. New filter-files can be created and tailored for use with any language or text file.

SAMPLE GXREF LISTING

GXREF sorts words using the ASCII collating sequence. A sample listing for file LIFE.C is shown next.

```
* * * * * * * * * Input Files * * * * * * * * *

life.c

* * * * * * * * * -X (Exclude) Words * * * * * * * * * *

for while do BEGIN END if char struct int long float extern printf fprintf
gets fgets puts fputs getc fgetc putc fputc open close fopen fclose FILE
break default return define case else endif exit ifdef sprintf read write
scanf strcpy strncpy strcmp strncmp switch toupper isdigit typedef ungetc
ungetch unlink unsigned

* * * * * * * * * -S (String Skip) * * * * * * * * * * *

Start string skip = /* Stop string skip = */
Start string skip = ' Stop string skip = '
Start string skip = " Stop string skip = "

* * * * * * * * * Miscellaneous * * * * * * * * *

Tab length = 8
Records are variable in length
Include file name signal = #include


COLS
  FILE -       life.c   48  111  127  167  169  173  211  211  215  222
                       245  247

COLS80
  FILE -       life.c   17

Del_char
  FILE -       life.c   31  179
```

The input-files section shows that file LIFE.C was input to GXREF. (Did I forget to mention that GXREF can handle multiple input-files? It can process a maximum of 50 input files.)

The excluded-word section lists those words marked as excluded (by the "-x" option) in the C filter.

The skip-string section contains those words listed after the "-s" option in the C filter.

More information is presented, and finally we find the cross-reference itself. The line

```
COLS
  FILE -       life.c   48  111  127  167  169  173  211  211  215  222
```

shows that COLS appears in LIFE.C on lines 48, 111, 127, etc. Line number 211 is repeated because COLS is used twice on that line.

```
if (world[row][COLS-1] >='X') add8(row,COLS-1); /* line 211 */
```

The macro statement

```
#define COLS    80      /*  this is line 17  */
```

generates

```
COLS80
  FILE -       life.c   17
```

so this use of COLS was not caught.

The next fragment shows how GXREF handles like-named arguments in different source-code files:

```
argc
  FILE -       later.c  16   17   25   30   38
  FILE -        dump.c   8    9   14
```

"Argc" appears on lines 16, 17, . . . in LATER.C and on lines 8, 9, and 14 in DUMP.C.

It would be useful at times to have the variable type listed next to each variable. For example,

```
tempx  - REAL*8
  FILE -       CIR2RP   10   11
```

The current release of GXREF doesn't support this feature. Perhaps a future version may.

In conclusion, GXREF can process a source file(s) at any stage of development, even if the file compiles with errors. An entire product or system can also be cross-referenced. GXREF will pick up embedded library names and their contents. A compiler won't do this. And at the maximum rate of 1000 cross-referenced lines per minute, GXREF won't slow you down either.

DIFS and LIKE

DIFS (price $139.00)
(DIFS Add-ons /A, /P, /L, $20.00 each)

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LIKE (price $40.00)

PC-DOS Version: DOS 2.x or later
Special Requirements: ANSI.SYS needed

There are four versions of the DIFS product: DIFS, DIFS/L, DIFS/A, and DIFS/P.

Each version finds and highlights the differences (change-cards) between two files, but the output takes a different format.

- DIFS — displays the change-cards on the screen or stores the screen display in a file.
- DIFS/L — creates change-card directives for use with LIKE (described later).
- DIFS/A — creates change-card directives for use with the mainframe ADR librarian.
- DIFS/P — creates change-card directives for use with the mainframe librarian PANVALET.

Let's look at DIFS/L and LIKE. DIFS/L finds changes between two files, say, OLD and NEW. These change-cards are generated in a format that LIKE can recognize. LIKE, which runs on the PC, combines NEW with the DIFS/L-generated change-cards to recreate OLD, so change-cards are used to reconstruct earlier versions of a source file.

There are several reasons why change-cards should be kept instead of complete files. First, only the updated version of the source file need be saved; older versions can be deleted since they may be recreated. This leads to less confusion and a smaller disk storage space requirement. Second, your one-line midnight change that destroyed the program can be identified and backed out since an audit trail exists.

Let's run through an example. Suppose we have two files, OLD and NEW. OLD is the original file and NEW is the updated version. We want to keep NEW; OLD will be deleted.

```
- - - - - - - - this is file OLD - - - - - - - -
this is line a
this is line b
this is line c
- - - - - - - - this is file OLD - - - - - - - -



- - - - - - - - this is file NEW - - - - - - - -
this is line b
this is line c
this is line a
- - - - - - - - this is file NEW - - - - - - - -
```

To find the differences and save them in file TEMP, use:

```
A>DIFS new old }temp
```

TEMP is shown below:

```
- - - DIFferenceS between NEW and OLD - - -
++this is line a
  this is line b
  this is line c
++this is line a
- - - - - - * * * * * * * * * * * * * - - - - - -
```

Additions and deletions are against NEW. The "++" lines indicate additions to NEW and the "—" lines deletions. These lines are shown highlighted on the screen. The second and third lines are the same in OLD and NEW.

To recreate OLD, add the first line (with the ++) to NEW, leave the second and third lines of NEW alone, and delete the fourth line of NEW.

To find the differences between OLD and NEW, and save them in OLD.DIF using a format that LIKE can recognize, type:

```
A>DIFS new old -l
```

Here's what OLD.DIF looks like with LIKE directives:

```
- - - - - - - this is file OLD.DIF - - - - - - -
@I O
this is line a
@D 3
- - - - - - - this is file OLD.DIF - - - - - - -
```

To recreate OLD using NEW and OLD.DIF, type:

```
A>LIKE new old
```

Note: DIFS and LIKE use the ANSI.SYS driver (provided on the PC-DOS 2.x diskette). If you don't use ANSI.SYS, escape sequences will appear on the screen.

SCIL (price $349.00)
PC-DOS Verson: DOS 2.x or later
Special Requirements ANSI.SYS needed

Demo Disk & User's Guide — (price $35.00)

SCIL is an interactive source-code librarian. It's the diamond in the SET tools package. Why?

Imagine you have written a FORTRAN program (it could happen to you!) called CIR2RP.FOR. A few days later you modify it. A decision must then be made. You can either delete the original CIR2RP.FOR and save the new version as CIR2RP.FOR, or you can rename CIR2RP.FOR to CIR2RP.BAK and save the new version in CIR2RP.FOR. More bugs will

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

appear later. What's the latest version of "CIR2RP" going to be called then? How about this: CIR2RP.BAK becomes CIR2RP.BA1, CIR2RP.FOR becomes CIR2RP.BAK, and the current version becomes CIR2RP.FOR. (You are no longer involved with programming, but with library management.) Suppose further that another programmer is also modifying his copy of CIR2RP.FOR at the same time that you're modifying your copy. How will you know if this is the case? Suppose even further that you want to maintain a copy of CIR2RP.FOR for the UNIX operating system and a copy for PC-DOS. A large portion of the FORTRAN code will remain the same, but there will be small, hard-to-find differences. How are you going to track all of that? On a ouija board? On scratch paper? Suggestion: Use SCIL.

Using SCIL you would create a library, say FORLIB, and a frame CIR2RP in this library. The CIR2RP frame will contain the history of CIR2RP.FOR from its original form to its current state.

Each frame is divided into levels. When the original source code of CIR2RP.FOR is PUT into frame CIR2RP, SCIL assigns it to "level 1." Later, if a bug is found in the level 1 code, GET is used to retrieve it into an ASCII file. You fix the bug using your own text editor and issue the PUT subcommand to save the updated source back into the CIR2RP frame. SCIL automatically assigns the changes between the original source and the new source to level 2.

This GET-old-level, fix bugs, PUT-new-level cycle is repeated when future bugs or enhancements are made to CIR2RP.FOR.

SCIL supports a maximum of 999 levels. In other words, SCIL does not support more than 999 PUTs to a single source frame.

Perhaps the next diagram will make these points clearer. It shows the library FORLIB, which contains at least three frames. Each frame contains the change history of a single source routine. The CIR2RP frame has three levels.

```
Library (=FORLIB)

   Frame  (=CIR2RP)                        Frame = (MO

   3  Level 3 = level 2 modifications       3  Level 2
   3  Level 2 = level 1 modifications       3  Level 1
   3  Level 1 = original CIR2RP.FOR


   Frame = (Next_file)
   3  Level 1 = original Next_file  3
```

Each level in a frame contains only the changes from the previous level. This economy allows the complete change history of a routine to be kept using a minimal amount of storage.

While SCIL can retrieve the source code in any level, it also offers more sophisticated features. Levels can be assigned RELEASE and VERSION labels and then manipulated using these labels. Another diagram will clarify this:

```
Frame  (=CIR2RP)

    3  VERSION = UNIX                    3
    3  Level 3 = level 2 modifications 3

    3  RELEASE = ALPHA                   3
    3  VERSION = PCDOS                   3
    3  Level 2 = original CIR2RP.FOR     3

    3  Level 1 = original CIR2RP.FOR     3
```

RELEASE labels mark the end of the source code in a release such as an ALPHA or BETA release. By using the same release label, say ALPHA, on levels in different FRAMES, SCIL can pluck out all of the many source files associated with the ALPHA release! That's exciting.

The discussion of levels and RELEASE labels had one underlying theme: that of incremental change to a base file. VERSION labels, on the other hand, are applied to levels that are fundamentally different from other non-labelled levels. For example, the VERSION labels PCDOS and UNIX are assigned to levels containing operating-system-specific code.

Let's move on and study how SCIL is actually used.

SCIL has five main subcommands: SET, PUT, GET, DISPLAY, and END. Each subcommand has several options, only a few of which are described here:

- SET — build a library; set VERSION and RELEASE labels
- PUT — put a frame into the specified library
- GET — get a frame from the specified library
- DIS — display levels, RELEASE & VERSION labels, comment journal, usage & metric tables
- END — return from SCIL to PC-DOS

Let's put SCIL through some paces.

Step 1:

BUILD the library FORLIB and add the empty frame CIR2RP to it. (Use the b(uild) obtion of SET subcommand.)

```
E>scil

Enter Sub-Command(s):set
Enter Library File Name:forlib
Enter Frame Name(s):cir2rp
Enter Option(s):b
Build of FORLIB.SCL library complete


SET PROCESSING -
Frame CIR2RP   Added in FORLIB.SCL library
SET COMPLETE.
```

Notice that the library FORLIB is contained in the file FORLIB.SCL.

The full list of prompts above shows that SCIL was used in the interactive mode. A command-line mode is also available. The equivalent build command-line is:

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
E>scil set FORLIB ,CIR2RP  -b
```

Step 2:

PUT the original source-code file CIR2RP.FOR into the frame CIR2RP in FORLIB.

```
E>scil put FORLIB ,CIR2RP -g

PUT PROCESSING -
Enter .GET filename for frame CIR2RP or ? for help:cir2rp.for
Enter Author Id:Administrator
Enter Comments for frame CIR2RP with <CR> only to end:
?CIR2RP converts a 2-d circle into a rational polynomial spline
?
...
Frame CIR2RP    from CIR2RP.FOR      PUT to FORLIB.SCL completed at level 1

PUT COMPLETE.
```

The "-g" option prompts the user for the file name to be placed in the frame. (This file is called a GET file because when SCIL retrieves a file it normally uses the default file extension of GET.)

A "?" is typed for the GET file if help is needed. On-line help is always available under SCIL.

The author's identification is "administrator." The administrator can type an unlimited-length comment describing the nature of this frame. This comment becomes part of a comment-journal which is viewed using the SCIL display option.

The three dots shown (. . .) beneath the "?" do not indicate an ommission on the reviewer's part. They are displayed, one at a time, to show that SCIL is running.

Finally, a confirmation message shows that the PUT operation was completed at level 1 — just as expected since CIR2RP.FOR is the only source code in this frame.

Step 3:

GET the source code from level 1 of frame CIR2RP into the file CIR2RP.FOR.

```
E>scil g FORLIB ,CIR2RP -g

GET PROCESSING -
Enter .GET filename for frame CIR2RP or ? for help:cir2rp.for
Enter 'O'wn or 'P'ermit if GET(s) for changes, else <CR>:p
Enter Author Id:Administrator
Begin GET of frame CIR2RP
...
Frame CIR2RP    from FORLIB.SCL GET to file CIR2RP.FOR      complete at level 1

GET COMPLETE.
```

The GET command is abbreviated to "g" (the first g) in the command line.

A GET file name of CIR2RP.FOR was typed. The code in level 1 will be placed in that file.

The Own and Permit keywords implement a form of file-locking. If you Own a file, then while others may retrieve working copies, SCIL will not allow them to retrieve a copy for making changes until your copy has been PUT back into the frame. Multiple users may get a frame using Permit-mode, modify the source, and PUT it back in any order. SCIL issues warning messages and a choice of actions if the changes overlap. These features are useful when a routine is being concurrently developed.

Step 4:

Using a text editor, we fix the bug found in the level 1 source (in file CIR2RP.FOR). Then we PUT the modified source back into frame CIR2RP. SCIL automatically assigns it to level 2 of the frame.

```
E>scil p FORLIB ,CIR2RP -g

PUT PROCESSING -
Enter .GET filename for frame CIR2RP or ? for help:cir2rp.for
Reget of CIR2RP    beginning
Reget complete.

Enter Comments for frame CIR2RP with <CR> only to end:
?Clarified relation between radius of circle and number of points in spline
?
Frame CIR2RP    from CIR2RP.FOR      PUT to FORLIB.SCL completed at level 2

PUT COMPLETE.
```

The PUT operation causes a reget — the level 1 file is retrieved and compared to the version being PUT into the frame. Only the changes are saved.

Step 5:

Display the metrics for the library FORLIB, using the metrics option (m).

```
E>scil d FORLIB ,CIR2RP -m

DIS PROCESSING -

 LIBRARY          BUILT      CHANGED  ACCESSED
 FORLIB.SCL
                04-10-85   04-10-85  04-10-85
 Metrics Report
 FRAME LEVEL RELEASE  VERSION  INSERT  DELETE   SAME
  NAME                          COUNT   COUNT   COUNT
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

CIR2RP   002                      6       5      66
         001                     71       0       0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

DIS COMPLETE.
```

The report shows the levels in frame CIR2RP, levels 1 and 2, and the number of lines inserted, deleted, and unchanged between levels. Level 1 contained 71 lines. Level 2 contains 72 (= 66 + 6) lines. (The DISplay "change" option lists the actual changes made to the updated file.)

Step 6:

Display the comment-journal, using the journal option (j).

```
E>scil d FORLIB ,CIR2RP -j

DIS PROCESSING -

 LIBRARY          BUILT      CHANGED  ACCESSED
 FORLIB.SCL
                04-10-85   04-10-85  04-10-85
 Journal Report
 FRAME LEVEL RELEASE  VERSION
  NAME
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
                    JOURNAL of COMMENTS
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

CIR2RP    002
Clarified relation between radius of circle and number of points in spline
          001
CIR2RP converts a 2-d circle into a rational polynomial spline
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

DIS COMPLETE.
```

Recognize those comments? They were typed by the administrator when the files were PUT into the frame.

Step 7:

Assume that we have created the new frame MODCHK, and that RELEASE label ONE was assigned to levels in CIR2RP and MODCHK. The DISplay option produces the following output.

```
 Tables Report
 FRAME LEVEL RELEASE  VERSION    DATE        AUTHOR
  NAME
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

CIR2RP    004 FOUR               04-11-85  Administrator
          003 THREE    MSDOS     03-20-85  Administrator
          002 TWO                03-05-84  Administrator
          001 ONE                06-11-82  Administrator
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

MODCHK    002          UNIX      03-20-85  Administrator
          001 ONE                06-11-82  Administrator
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

DIS COMPLETE.

E>ecil g FORLIB  ,CIR2RP,MODCHK  -rONE
```

The r(elease) option was used to get the files — from frames CIR2RP and MODCHK — that have a RELEASE label of ONE. These files are placed in CIR2RP.GET and MODCHK.GET.

In the table-report, level 3 has the MSDOS VERSION label. When you GET a file at level 4, it does not reflect changes made to the MSDOS version. This makes sense, as the MSDOS version is related only to other MSDOS-labelled levels.

CLOSING THOUGHTS

While this review emphasized the source-code management aspect of SCIL, SCIL can also manage programmer documentation or any type of ASCII file that requires updating.

The user's manual is comprehensive but overwhelming because of the many SCIL options. For beginners, I suggest studying Appendix A. It contains an index of all SCIL subcommands and options. This two-page index is a summary of functions. Then read the first three chapters of the manual (introduction + tutorial) and dive into experimenting with SCIL. SCIL is a predictable, bug-free product (haven't found any yet) that will train you as well as any manual. The manual contains helpful explanations of error and warning messages, so you shouldn't feel lost at any time.

CONCLUSION

The SET tools are billed as programmer productivity tools. They live up to this claim, being well crafted in both design and implementation.

You will find applications for any of these tools, but SCIL deserves special mention. It brings sophisticated library management, the missing link, to your PC. □

A FULL C COMPILER FOR $49.95

The Eco-C88 C compiler is setting a new standard for price and performance. Compare Eco-C88's performance to compilers costing up to 10 times as much:

	Seive	Fib	Defref	Matrix
Execute	12.1 sec.	43.1 sec.	13.7 sec.	21.3 sec.
Code Size	7782	7754	7772	9120
Compile-link	76 Sec.	77 Sec.	77 Sec.	92 Sec.

Eco-C88 Rel. 2.20, on IBM PC with 2 floppy disks, 256K. Benchmarks from Feb., 1985 **Computer Language.**

Eco-C88 includes:
* All operators and data types (except bit fields)
* Error messages in English with page numbers that reference the **C Programming Guide** - a real plus if you're just getting started in C.
* Over 170 library functions, including color and transcendentals
* New Library functions for treating memory as a file
* User-selectable ASM or OBJ output (no assembler required)
* 8087 support with 8087 sensed at *runtime*
* cc and "mini-make" for easy compiles (with source)
* Fast, efficient code for all IBM-PC, XT, AT and compatibles using MSDOS 2.1 or later.
* Complete user's manual

If ordered with the compiler, the C library source code (excluding transcendentals) is $10.00 and the ISAM file handler (as published in the **C Programmer's Library,** Que Corp) in OBJ format is an additional $15.00. Please add $4.00 for shipping and handling. To order, call or write:

MasterCard VISA AMERICAN EXPRESS

Ecosoft Inc.
6413 N. College Avenue
Indianapolis, IN 46220
(317) 255-6476 • 8:30-4:30

ECOSOFT INC.

Eco-C (Ecosoft), MSDOS (Microsoft), UNIX (Bell Labs), CP/M (Digital Research), Z80 (Zilog), 8086, 8087, 8088 (Intel).

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ARE YOU AN ISSUE SHY OF A FULL SET?

- Vol. 1, No. 1: Norton Pascal I/O, Program Style Manual, COBOL, System Analysis, BASIC modules, Rounding error.
- Vol. 1, No. 2: Defining character sets, C86 BIOS calls, Copy protection, Pascal code, Input routines, Writing an Assembler.
- Vol. 1, No. 3: ASM dumb terminal, C source code, ASM graphics line drawing, Text editors reviewed, Utilities.
- Vol. 1, No. 4: Diskette directory via Pascal, MASM WHILE-WEND, C/ASM interface, ASM stay resident routine, Structured programming.
- Vol. 1, No. 5: Multi-tasking software review, Transporting C Programs, Programmers calculator, dBase bargraphs, Utilities.
- Vol. 1, No. 6: ASM printer configuration, DOS patch utility, Hi-speed graphics, Software ownership, dWINDOW review.
- Vol. 2, No. 1: Freelance consulting, Sorts in C, Pascal text editor source code, Designing national database, Ultra utilities.
- Vol. 2, No. 2: Understanding C through Assembly language, Utility routines in C and Assembly, Music in C (using a C music program), Cornucopia of software development tools.
- Vol. 2, No. 3: Software development environments, Creating a productive screen handling tool, Generating graphics within windows, Create, edit and play your musical ideas, Linking Assembler to Pascal and Basic.
- Vol. 2, No. 4: Writing a custom ROM BIOS for PC-DOS, The future of disk protection, Time-saving MASM Macros, C routines you can purchase.
- Vol. 2, No. 5: Scroll control using BASIC and Assembler, RTool, a PC-resident programmer's toolkit, Calling Assembly routines from dBase II, Pascal's pointer power.
- Vol. 2, No. 6: Periscope-Building a hardware board for the IBM PC, AT&T surprises all, dBase II to dBase III?, Intel 80286 bugs by FNE, calendar functions for BASIC.
- Vol. 3, No. 1: November's Comdex, C - its uses and abuses, Using the C preprocessor, Memory segmentation examples, Index files (part 1), Stuctured programming language, Periscope, a symbolic debugger.
- Vol. 3, No. 2: UNIX! (again), Artificial intelligence, Doctor C's tips — pointers to functions, Calendar date routines in BASIC, Calculate record layouts, automatically!
- Vol. 3, No. 3: NAPLPS and the professional programmer, Instant-C, Printer control from a batch file, Repositioning the stars in Starfinder, Reducing EXE file size through data positioning, tools for better programming.

Name _____

Company _____

Address _____

City _____ State _____ Zip _____

Telephone _____

Send me the following back issues: (Circle the issue(s) desired)
$5.50 each U.S., ($6.50 Canada, $8.50 Foreign, U.S. funds)
PRE-PAID ONLY

Vol. 1, No. 1	Vol. 1, No. 5	Vol. 2. No. 3	Vol. 3, No. 1
Vol. 1, No. 2	Vol. 1, No. 6	Vol. 2. No. 4	Vol. 3, No.2
Vol. 1, No. 3	Vol. 2, No. 1	Vol. 2. No. 5	Vol. 3, No. 3
Vol. 1, No. 4	Vol. 2, No. 2	Vol. 2. No. 6	

Total amount of order _____

☐ Check enclosed

☐ Charge my Bankcard: ☐ VISA ☐ MC

_____ Expir. Date _____

Signature _____

Send to:
PROGRAMMER'S JOURNAL
P.O. Box 30160
Eugene, OR 97403
(503) 484-2162

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C + C UTILITY LIBRARY = PRODUCT

• We have over *200 complete, tested, and, documented functions.* All source code and demo programs are included.

• The library was specifically designed for software development on the IBM PC, XT, AT and compatibles. There are *no royalties.*

• Over 95% of the source code is written in C. *Experienced programmers* can easily "customize" functions. *Novices* can learn from the thorough comments.

We already have the functions you are about to write

Concentrate on software development—not writing functions.

THE C UTILITY LIBRARY includes:

• Best Screen Handling Available • Windows • Full Set of Color Graphics Functions • Better String Handling Than Basic • DOS Directory and File Management • Execute Programs, DOS Commands and Batch Files • Complete Keyboard Control • Extensive Time/Date Processing • Polled ASYNC Communications • General DOS/BIOS gate • And More •

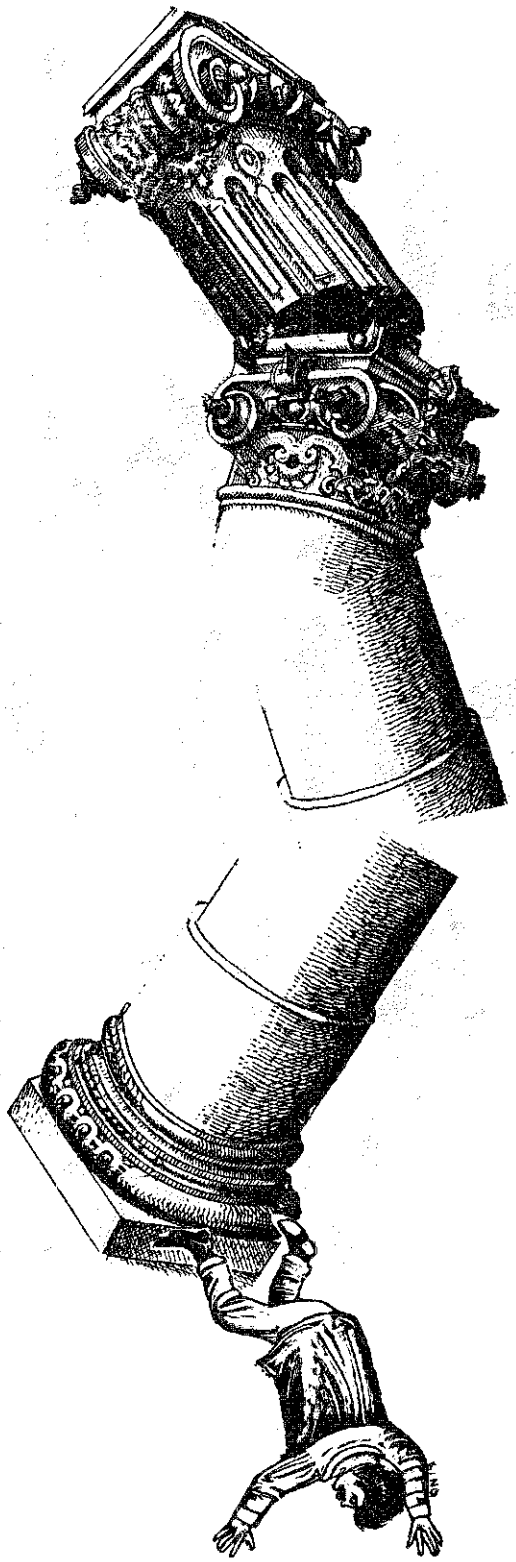
• The Library is compatible with: Lattice, Microsoft, Computer Innovations, Mark Williams and DeSmet.

C Compilers: Lattice C—$349, Computer Innovations C86—$329; Mark Williams C—$449.

C UTILITY LIBRARY $149

Order direct or through your dealer. *Specify compiler* when ordering. Add $4.00 shipping for UPS ground, $7.00 for UPS 2-day service. NJ residents add 6% sales tax. Master Card, Visa, check or P.O.

ESI ESSENTIAL SOFTWARE, INC
P.O. Box 1003 Maplewood, New Jersey 07040 914/762-6605



Famous Computer Personalities #68: Fralick Pittore suffering severe injuries sustained while trying to insert a column on an early version of a spreadsheet.

1	2
3	4

Have we got your number?

Is Programmer's Journal reaching you at the correct address? Please notify us of any address change or correction.

Is it nearing time to renew? Your Programmer's Journal mailing label shows when your subscription expires. Renew early to avoid any interruption in your subscription service!

Programmer's Journal, P.O. Box 3000, Dept. EE, Denville, NJ 07834

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LATTICE C USERS COME TOGETHER

Lattice, Inc. will give Bill Hunt, author of The C Toolbox, their full support as an independent Lattice C Users Group comes together under his guidance. The group will serve users of Lattice, Lifeboat, and Microsoft versions. For a $30.00, one-year membership fee, members receive a sixteen-page bi-monthly newsletter that explains C features, provides examples of their use, explores new ways to use Lattice C, and includes a question and answer column.

The newsletter will be supplemented by a disk containing source files, demo programs, library functions, plus the best of the public-domain C programs. You don't have to be a Lattice C user to join.

Lattice C Users' Group
P.O. Box 271965
Concord, CA 94527

For registered users of the Lattice C Compiler, Lattice Cross Compilers, and other Lattice products, Lattice, Inc. offers *Lattice Works*, a quarterly newsletter featuring current revision numbers of Lattice programs and advance notice of new products. Complete and return the registration card enclosed with the product or write for a free sample issue.

Lattice, Inc.
P.O. Box 3072
Glen Ellyn, IL 60138
(312) 858-7950
TWX 910-291-2190

UPGRADE 8088 PRODUCTS TO THE 80286

Edsun Labs has announced the EL286-88 Processor Converter. The VSLI circuit permits easy upgrade of 8088 products to the new 80286 processor. Steven D. Edelson, president of Edsun Labs, welcomes technical questions from software developers.

Edsun Laboratories, Inc.
7 Sears Road
Wayland, MA 01778
(617) 358-5667

MINI CAPABILITY ON A MICRO

From California Software Products, Inc., the BABY/36 is a complete software system that allows use of IBM System/36 RPG II programs on IBM PCs and some compatibles. BABY/36 allows the user to download RPG II source code and recompile it on a micro.

The BABY/36 consists of nine components: (1) RPG II Compiler, (2) Operation Control Language Processor, (3) Screen Format Generator, (4) Source Entry Utility, (5) Sort Utility, (6) RPG Program Control, (7) Data Exchange Utility, (8) Data File Utility, and (9) Screen Design Aid. Other utilities included are (1) Data File Management, (2) Library Member Management, (3) System Management, and (4) Other Compilers and Generators.

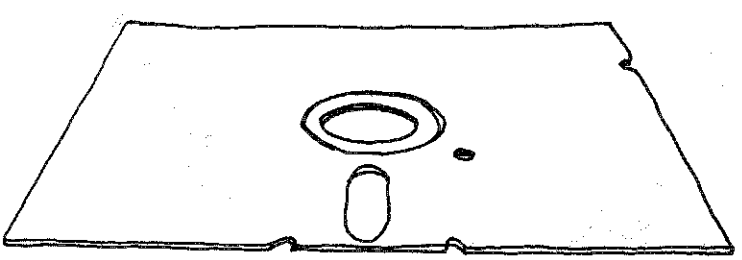
Minimum system requirements are an IBM PC, PC/XT, PC/AT, PC/3270, or some compatibles; 384K memory; hard disk; one 5 1/4" disk drive, DOS 3.0 or above. The price for the BABY/36 RPG II Stand Alone System is $3,000 or $3,500 with the Data File Utility. An optional Run Time Support System is available for $700, or with Data File Execution for $875. Maintenance is free for one year.

California Software Products, Inc.
525 North Cabrillo Park Drive
Suite 300
Santa Ana, CA 92701
(714) 973-0440
Telex 685645

CONCURRENT DOS-286 AVAILABLE NOW

Concurrent DOS-286 from Digital Research, Inc. has been developed specifically for Intel's 80286 microprocessor. The new operating system offers features that can be used for networking, communications, multi-tasking, and single- or multi-user systems. Concurrent DOS-286 allows non-IBM look-alike 286-based systems to run all PC DOS as well as CP/M 86 applications. It will allow users to run several applications at one time and provide record and file locking features. DOS-286 is currently targeted at the IBM PC/AT.

Intel
3065 Bowers
Santa Clara, CA 95051



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

TRY BEFORE YOU BUY

Software Concepts will let you try their new Computerized Atlas before you buy. Send $5 ($1 for the program and $4 for postage, handling, and a Verbatim DataLife disk). They'll send you a copy of the program that you can run three times, for as long as you like each time, and then the program locks. If you like the program, you call them toll-free to order and they'll give you the unlocking code. If you decide you don't want the program, format the disk, and have a blank.

The atlas features a full-color, three-dimensiional globe; magnifying capabilities; fact sheets on population, size, capital, and monetary units; and facts about over 2,500 cities. The atlas will also compute the distance between places in land miles, nautical miles, or kilometers. □

Software Concepts
P.O. Box 3323
Wallingford, CT 06494

C Programmers Quit Working So Hard!

THE GREENLEAF FUNCTIONS™

***The* GREENLEAF FUNCTIONS GENERAL LIBRARY** has over 200 functions in C and assembler. Strength in DOS, video, string, printer, async, and systems interface. All DOS 1 and 2 functions are in assembler for speed. All video capabilities of PC supported. All printer functions. 65 string functions. Extensive time and date. Directory searches. Polled mode async. (If you want interrupt driven, ask us about the **Greenleaf Comm Library.**) Function key support. Diagnostics. Rainbow Color Text series. Much, much more. **The Greenleaf Functions.** Simply the finest C library (and the most extensive). All ready for you.

THE GREENLEAF FUNCTIONS™

The Library of C Functions that probably has just what you need . . . **TODAY!**

- already has what you're working to re-invent
- already has over 200 functions for the IBM PC, XT, AT, and compatibles
- already complete . . . already tested . . . on the shelf
- already has demo programs and source code
- already compatible with all popular compilers
- already supports all memory models, DOS 1.1, 2.0, 2.1
- already optimized (parts in assembler) for speed and density
- already in use by thousands of customers worldwide
- already available from stock (your dealer probably has it)
- It's called the **GREENLEAF FUNCTIONS.**

The Library of C Functions is Waiting for You

Specify compiler when ordering. Add $7.00 for UPS second-day air (or $5.00 for ground). Texas residents add sales tax. Mastercard, VISA, check or P.O. In stock, shipped same day.

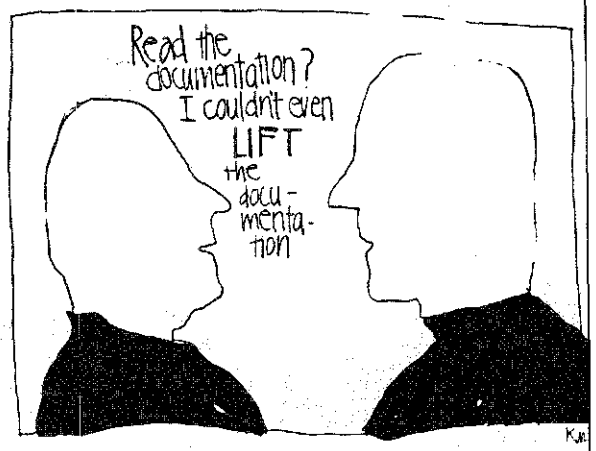
General Libraries	$185
Comm Library	$185
CI C86 Compiler	$349
Lattice C	$395
Mark Williams	$475

For information: 214-446-8641

Prices are subject to change without notice.

GREENLEAF SOFTWARE

2101 HICKORY DR.
CARROLLTON, TX 75006



ADVERTISERS INDEX JULY/AUGUST 1985

Baen Books 9
Blaise Computing 4
Borland International . . . . . Cover/Back
Computer Faire, Inc. . . . . Cover/Inside Back
Cybernetic Micro Systems 14
Data Base Decisions 35
Ecosoft, Inc. 44
Essential Software 46
Greenleaf Software 48
Lifeboat Associates . . . . . Cover/Inside Front
MicroMotion 6
Morgan Computing 19
OES Systems 14
Phoenix Computer Products 24-25
Programmers Shop 7
Qualitas 31
Rational Systems, Inc. 11
J.L. Schuller & Associates 8
Software Horizons 11
Sterling Castle 35
STSC 3

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Thousands of computers are gathering in San Francisco & Boston

3rd Personal Computer Faire

Sept. 5–7
Civic Auditorium & Brooks Hall
SAN FRANCISCO

THE 8TH NORTHEAST COMPUTER FAIRE

Sept. 26–29
Bayside Exposition Center
BOSTON

You won't want to miss these great shows!

- **Thousands of Computer Products** exhibited by hundreds of leading and innovative companies, including IBM, Apple, and many others.
- **Special Areas For "Hands-On" Demos.**
- **A Comprehensive Conference Program** (included in admission) attracting industry speakers and computer users from business, education, government and scientific, engineering & research communities.
- **Professional Development Seminars.**
- **Free One-On-One Consultations.**
- **Incredible "Show Pricing"** on all kinds of products.

At this first-rate event, you can comparison shop for all the leading personal computers, add-ons and software, question factory experts about the capabilities of their machines, participate in computer demonstrations and take in a conference program that includes over 40 sessions and panels. If you decide to buy at the show, you'll save hundreds, even thousands of dollars; everything on display is for sale at super show prices!

CORPORATE OFFICE
Computer Faire, Inc.
181 Wells Avenue
Newton, MA 02159

WEST COAST OFFICE
Computer Faire, Inc.
611 Veterans Boulevard
Redwood City, CA 94063

An exclusive production of **Computer Faire, Inc.**/A Prentice-Hall Company

Please send me information on attendee preregistration discounts.

Name _____

Company _____

Street _____

City/Town _____ State _____ Zip _____

Phone _____

I would like to ☐ Attend ☐ Exhibit
☐ 3rd Personal Computer Faire
☐ 8th Northeast Computer Faire

Send this coupon to:
Computer Faire, Inc.
181 Wells Avenue
Newton, MA 02159
You can call us at **(617) 965-8350.**

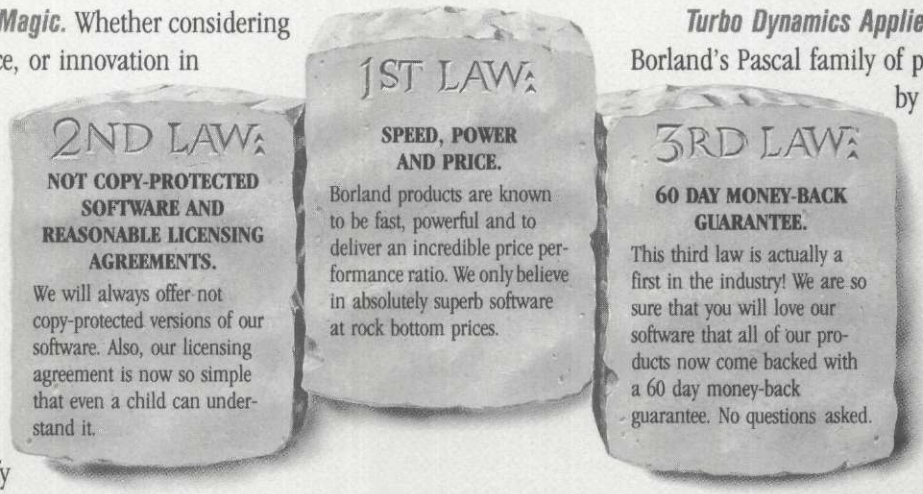
PJ

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PC WEEK'S PRODUCT OF THE YEAR
PC MAGAZINE'S AWARD FOR TECHNICAL EXCELLENCE

Borland Introduces the Laws of *TURBO DYNAMICS*™

Laws That Work Like Magic. Whether considering technological excellence, or innovation in areas such as pricing, not copy-protection, licensing agreements, site licenses, 60 day money-back guarantee —Borland is clearly recognized as the software industry leader. The following three laws of "*Turbo Dynamics*"™ exemplify our pledge for excellence.


2ND LAW:
NOT COPY-PROTECTED SOFTWARE AND REASONABLE LICENSING AGREEMENTS.
We will always offer not copy-protected versions of our software. Also, our licensing agreement is now so simple that even a child can understand it.
1ST LAW:
SPEED, POWER AND PRICE.
Borland products are known to be fast, powerful and to deliver an incredible price performance ratio. We only believe in absolutely superb software at rock bottom prices.
3RD LAW:
60 DAY MONEY-BACK GUARANTEE.
This third law is actually a first in the industry! We are so sure that you will love our software that all of our products now come backed with a 60 day money-back guarantee. No questions asked.

Turbo Dynamics Applies to Turbo Pascal. Borland's Pascal family of products is growing by leaps and bounds. You can now join hundreds of thousands of users and enter the world of Turbo Pascal programming. And remember, all three laws of *Turbo Dynamics* apply to all Borland products.

TURBO PASCAL™ $69.95

The industry standard. With more than 350,000 users worldwide Turbo Pascal is the industry's de facto standard. Turbo Pascal is praised by more engineers, hobbyists, students and professional programmers than any other development environment in the history of microcomputing. And yet, Turbo Pascal is simple and fun to use. **Free spreadsheet** included on every Turbo disk with ready-to-compile source code. ***Options:*** We offer the exciting Binary Coded Decimal (BCD) option for your business applications as well as an 8087 option for your number-crunching applications at a very low charge. Please refer to the coupon. ***Portability.*** Turbo Pascal is available today for most computers running PC-DOS, MS-DOS, CP/M-80 or CP/M-86. ***Jeff Duntemann, PC Magazine:*** "In its simplicity it achieves an elegance that no other language compiler has ever displayed."

TURBO GRAPHIX TOOLBOX™ $54.95

High resolution monochrome graphics and window management for the IBM PC. The Turbo Graphix Toolbox will give even a beginning programmer the expert's edge. It's a complete library of Pascal procedures and functions. Tools that will allow you to draw and hatch pie charts, bar charts, circles, rectangles and a full range of geometric shapes. Procedures that will save and restore graphic images to and from disk. And much, much, more. You may incorporate part or all of these tools in your programs and yet we won't charge you any royalties. Best of all, these functions and procedures come complete with commented source code on disk ready to compile.

TURBO TUTOR™ $34.95

From start to finish in 300 pages. Turbo Tutor is for everyone from novice to expert. Even if you've never programmed before Turbo Tutor will get you started right away. ***A must.*** You'll find the source code for all the examples in the book on the accompanying disk ready to compile. Turbo Tutor might be the only reference on Pascal and programming you'll ever need.

TURBO DATABASE TOOLBOX™ $54.95

The Turbo Database Toolbox is the perfect complement to Turbo Pascal. It contains a complete library of Pascal procedures that allows you to sort and search your data and build powerful applications. It's another Borland set of tools that will give the beginning programmer the expert's edge. ***Get started right away: free database!*** Included on every Toolbox disk is the source code to a working data base which demonstrates how powerful and easy to use our search system, Turbo-Access, really is. Modify it to suit your individual needs or just compile it and run. ***Remember, no royalties!***

NOT COPY-PROTECTED

TURBO PASCAL FAMILY

Available at better dealers nationwide. Call (800) 556-2283 for the dealer nearest you. To order by Credit Card call (800) 255-8008, CA (800) 742-1133

Carefully Describe your Computer System!

Mine is: ___ 8 bit ___ 16 bit
I Use: ___ PC-DOS ___ MS-DOS
___ CP/M 80 ___ CP/M 86

My computers' name/model is:

The disk size I use is:
☐ 3½" ☐ 5¼" ☐ 8"

Pascal 3.0	$ 69.95
Pascal w/8087	$109.90
Pascal w/BCD	$109.90
Pascal w/8087 & BCD	$124.95
Turbo Database Toolbox	$ 54.95
Turbo Graphix	$ 54.95
Turbo Tutor	$ 34.95

60 DAY MONEY-BACK GUARANTEE

*These prices include shipping to all U.S. cities. All foreign orders add $10 per product ordered.

Amount: (CA 6% tax) _____

Payment: VISA MC BankDraft Check

Credit Card Expir. Date: ___/___

Card #: _____

Name: _____

Shipping Address: _____

City: _____

State: _____ Zip: _____

Telephone: _____

COD's and Purchase Orders WILL NOT be accepted by Borland. California residents: add 6% sales tax. Outside USA: add $10 and make payment by bank draft, payable in US dollars drawn on a US bank.

F36

BORLAND INTERNATIONAL

4585 Scotts Valley Drive, Scotts Valley CA 95066
Phone (408) 438-8400 Telex 172373

Copyright 1985 Borland International BI-1011

Turbo Pascal, Turbo Database Toolbox, Turbo Graphix Toolbox, Turbo Tutor and Turbo Dynamics are trademarks of Borland International, Inc.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED